PROJECT REPORT

DATA SCIENCE AND ANALYTICS

Name: Suchit Meshram
Batch: Data Science &
Analytics
Submitted to: karan sir

INDEX

Machine Learning : Predict whether the customer will be a defaulter or not.

About the project

The dataset contains information about applicants. Banks, globally, use this kind of dataset and type of informative data to create models to help in deciding on who is defaulter/not for a loan.

After all the exploratory data analysis, cleansing and dealing with all the anomalies we might (will) find along the way, the patterns of a who is fraud or not applicant will be exposed to be learned by machine learning models.

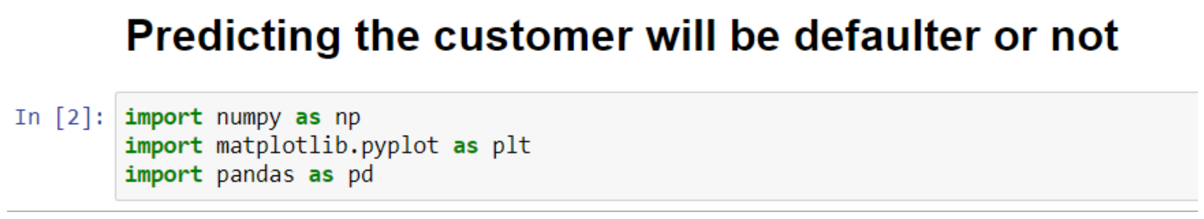
Project structure

The project divides into three categories:
1.EDA: Exploratory Data Analysis
2.Data Wrangling: Cleansing and Feature Selection
3.Machine Learning: Predictive Modelling

The dataset

Fraud dataset

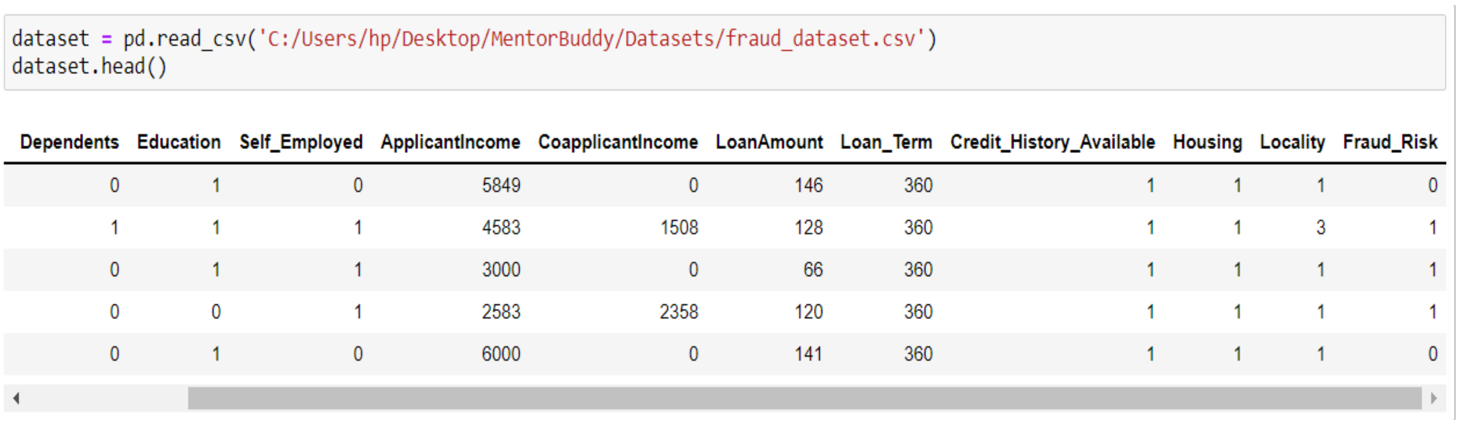
Importing the libraries and dependencies required:



Predicting the customer will be defaulter or not

```
In [2]: import numpy as np
        import matplotlib.pyplot as plt
        import pandas as pd
```

Loading and displaying the dataset:

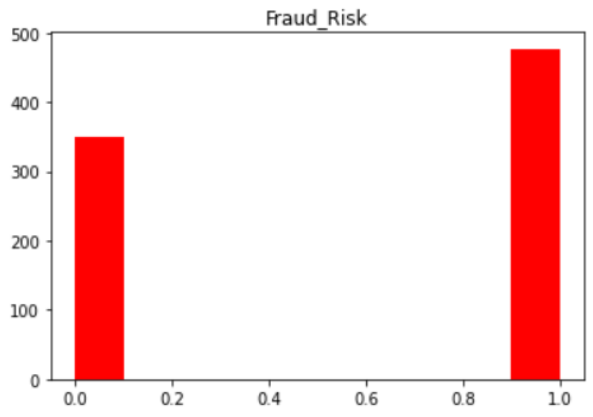


```
dataset = pd.read_csv('C:/Users/hp/Desktop/MentorBuddy/Datasets/fraud_dataset.csv')
dataset.head()
```

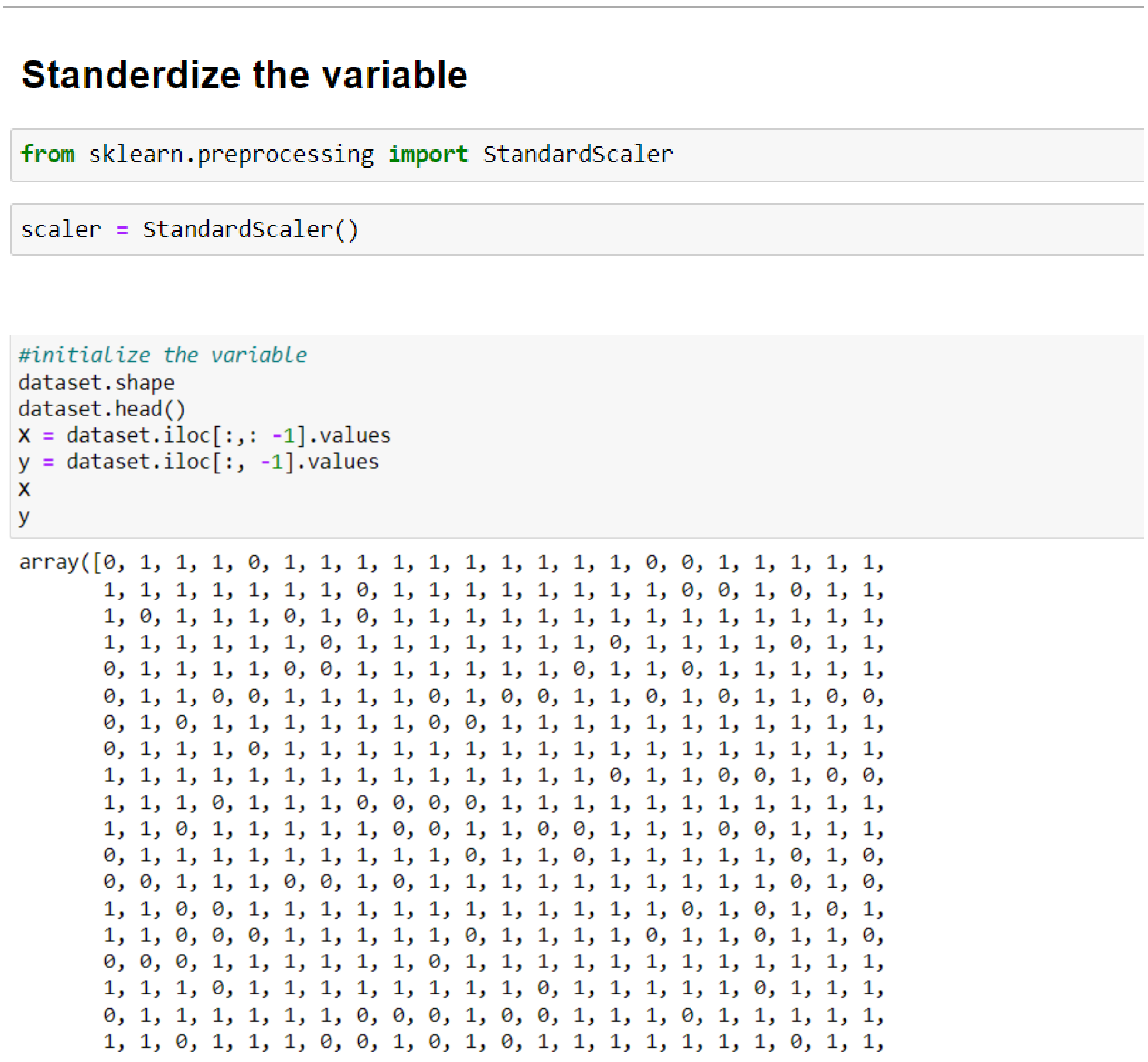
Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Term	Credit_History_Available	Housing	Locality	Fraud_Risk
0	1	0	5849	0	146	360	1	1	1	0
1	1	1	4583	1508	128	360	1	1	3	1
0	1	1	3000	0	66	360	1	1	1	1
0	0	1	2583	2358	120	360	1	1	1	1
0	1	0	6000	0	141	360	1	1	1	0

Data Visualization

```
In [12]: dataset.hist(column="Fraud_Risk",grid=False, color = "red")

Out[12]: array([[<AxesSubplot:title={'center':'Fraud_Risk'}>]], dtype=object)
```



Standerdizing the variable:



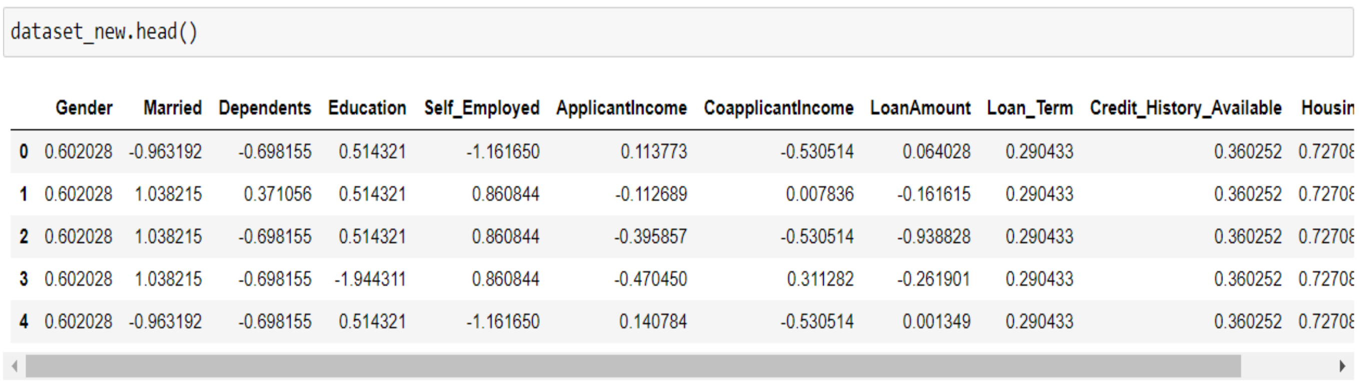
Standerdize the variable

```
from sklearn.preprocessing import StandardScaler
```

```
scaler = StandardScaler()
```

```
#initialize the variable
dataset.shape
dataset.head()
X = dataset.iloc[:,: -1].values
y = dataset.iloc[:, -1].values
X
y
```

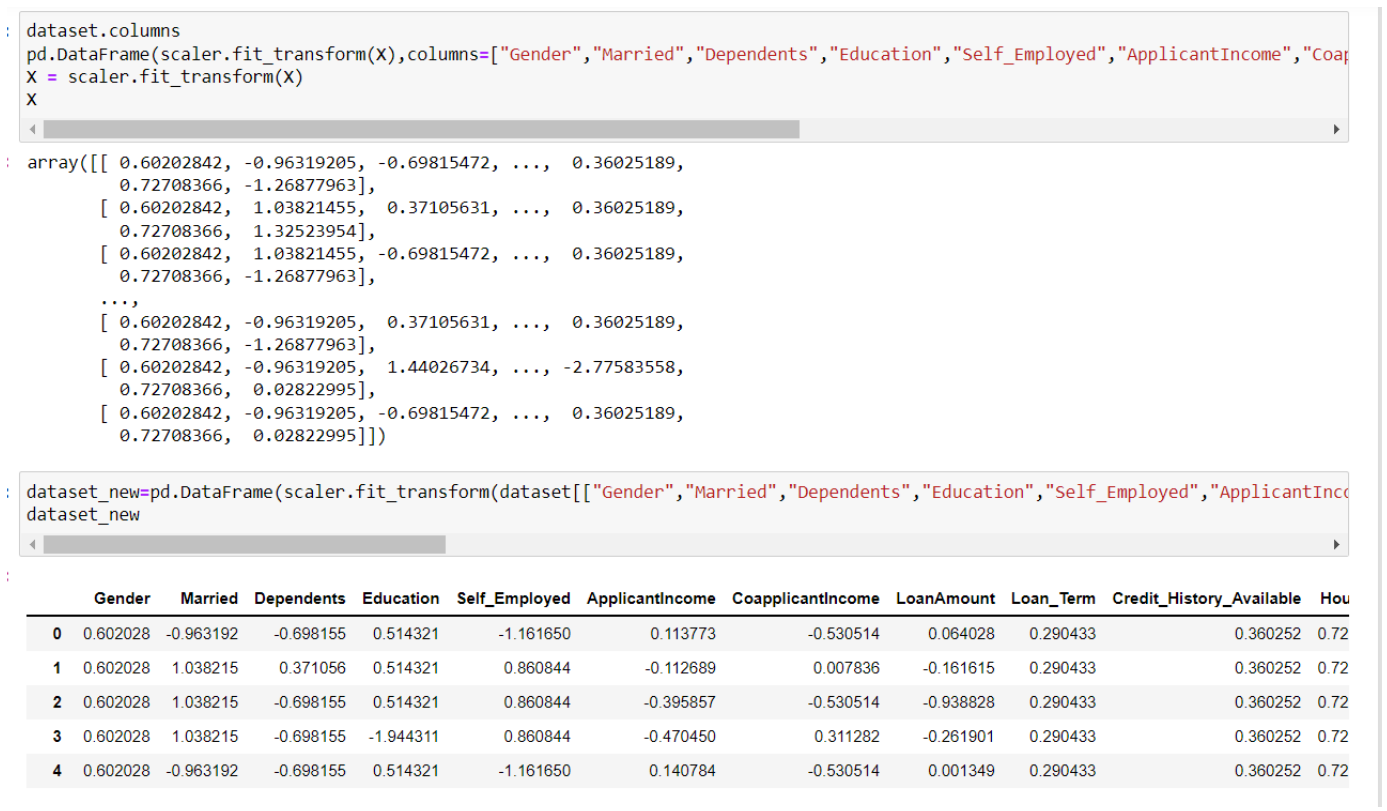
```
array([0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1,
       1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1,
       0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1,
       0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0,
       0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0,
       1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1,
       0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0,
       0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0,
       1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1,
       1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0,
       0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1,
       0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1,
       1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1,
```



```
dataset_new.head()
```

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Term	Credit_History_Available	Housin
0	0.602028	-0.963192	-0.698155	0.514321	-1.161650	0.113773	-0.530514	0.064028	0.290433	0.360252	0.72708
1	0.602028	1.038215	0.371056	0.514321	0.860844	-0.112689	0.007836	-0.161615	0.290433	0.360252	0.72708
2	0.602028	1.038215	-0.698155	0.514321	0.860844	-0.395857	-0.530514	-0.938828	0.290433	0.360252	0.72708
3	0.602028	1.038215	-0.698155	-1.944311	0.860844	-0.470450	0.311282	-0.261901	0.290433	0.360252	0.72708
4	0.602028	-0.963192	-0.698155	0.514321	-1.161650	0.140784	-0.530514	0.001349	0.290433	0.360252	0.72708

The dataset has 829 observations and 13 variables including the target, divided into 13 numeric and 0 categoric features.



```
dataset.columns
pd.DataFrame(scaler.fit_transform(X),columns=["Gender","Married","Dependents","Education","Self_Employed","ApplicantIncome","Coap
X = scaler.fit_transform(X)
X
```

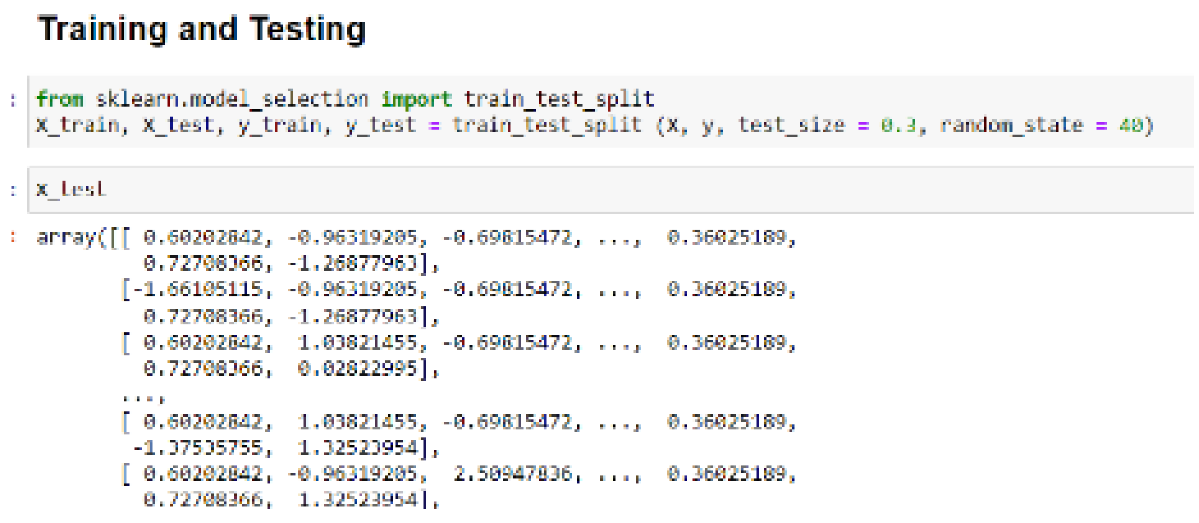
```
array([[ 0.60202842, -0.96319205, -0.69815472, ...,  0.36025189,
         0.72708366, -1.26877963],
       [ 0.60202842,  1.03821455,  0.37105631, ...,  0.36025189,
         0.72708366,  1.32523954],
       [ 0.60202842,  1.03821455, -0.69815472, ...,  0.36025189,
         0.72708366, -1.26877963],
       ...,
       [ 0.60202842, -0.96319205,  0.37105631, ...,  0.36025189,
         0.72708366, -1.26877963],
       [ 0.60202842, -0.96319205,  1.44026734, ..., -2.77583558,
         0.72708366,  0.02822995],
       [ 0.60202842, -0.96319205, -0.69815472, ...,  0.36025189,
         0.72708366,  0.02822995]])
```

```
dataset_new=pd.DataFrame(scaler.fit_transform(dataset[["Gender","Married","Dependents","Education","Self_Employed","ApplicantInc
dataset_new
```

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Term	Credit_History_Available	Hou
0	0.602028	-0.963192	-0.698155	0.514321	-1.161650	0.113773	-0.530514	0.064028	0.290433	0.360252	0.72
1	0.602028	1.038215	0.371056	0.514321	0.860844	-0.112689	0.007836	-0.161615	0.290433	0.360252	0.72
2	0.602028	1.038215	-0.698155	0.514321	0.860844	-0.395857	-0.530514	-0.938828	0.290433	0.360252	0.72
3	0.602028	1.038215	-0.698155	-1.944311	0.860844	-0.470450	0.311282	-0.261901	0.290433	0.360252	0.72
4	0.602028	-0.963192	-0.698155	0.514321	-1.161650	0.140784	-0.530514	0.001349	0.290433	0.360252	0.72

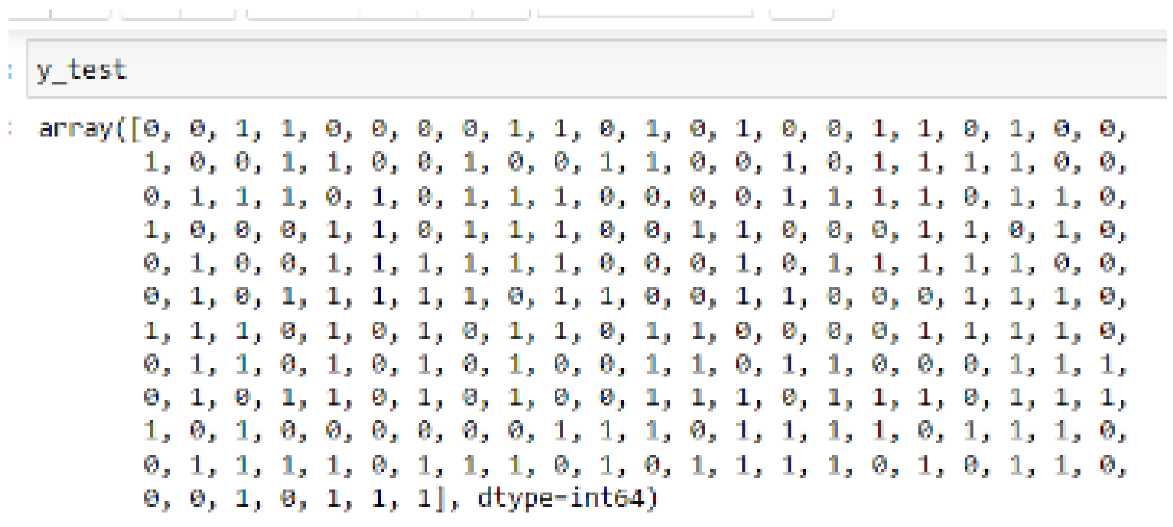
Splitting the data in training and testing sets

The first step when building a model is to split the data into two groups, which are typically referred to as training and testing sets. The training set is used by the machine learning algorithm to build the model. The test set contains samples that are not part of the learning process and is used to evaluate the model's performance. It is important to assess the quality of the model using unseen data to guarantee an objective evaluation.



Training and Testing

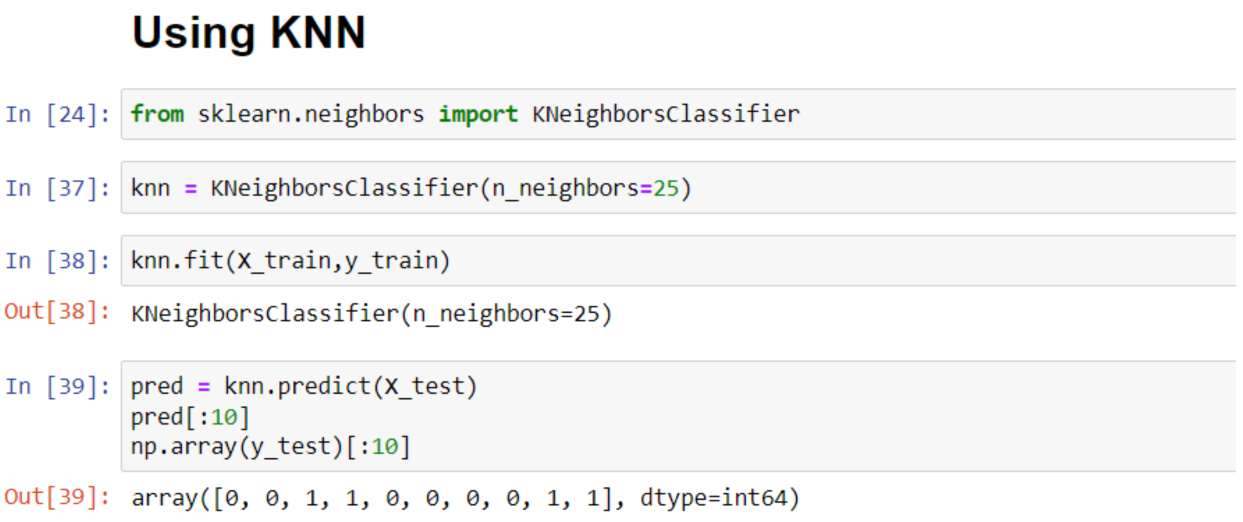
```
: from sklearn.model_selection import train_test_split
  X_train, X_test, y_train, y_test = train_test_split (X, y, test_size = 0.3, random_state = 40)

: X_test

: array([[ 0.60202842, -0.96319205, -0.69815472, ...,  0.36025189,
           0.72708366, -1.26877963],
         [-1.66105115, -0.96319205, -0.69815472, ...,  0.36025189,
           0.72708366, -1.26877963],
         [ 0.60202842,  1.03821455, -0.69815472, ...,  0.36025189,
           0.72708366,  0.02822995],
         ...,
         [ 0.60202842,  1.03821455, -0.69815472, ...,  0.36025189,
          -1.37535755,  1.32523954],
         [ 0.60202842, -0.96319205,  2.50947836, ...,  0.36025189,
           0.72708366,  1.32523954],
```



```
: y_test

: array([0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0,
         1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0,
         0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0,
         1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0,
         0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0,
         0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0,
         1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0,
         0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1,
         0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1,
         1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0,
         0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0,
         0, 0, 1, 0, 1, 1, 1], dtype=int64)
```

Assessing multiple algorithms

Algorithm selection is a key challenge in any machine learning project since there is not an algorithm that is the best across all projects. Generally, we need to evaluate a set of potential candidates and select for further evaluation those that provide better performance, here I'm using KNN algorithm.



Using KNN

```
In [24]: from sklearn.neighbors import KNeighborsClassifier

In [37]: knn = KNeighborsClassifier(n_neighbors=25)

In [38]: knn.fit(X_train,y_train)

Out[38]: KNeighborsClassifier(n_neighbors=25)

In [39]: pred = knn.predict(X_test)
         pred[:10]
         np.array(y_test)[:10]

Out[39]: array([0, 0, 1, 1, 0, 0, 0, 0, 1, 1], dtype=int64)
```

Evaluation metrics

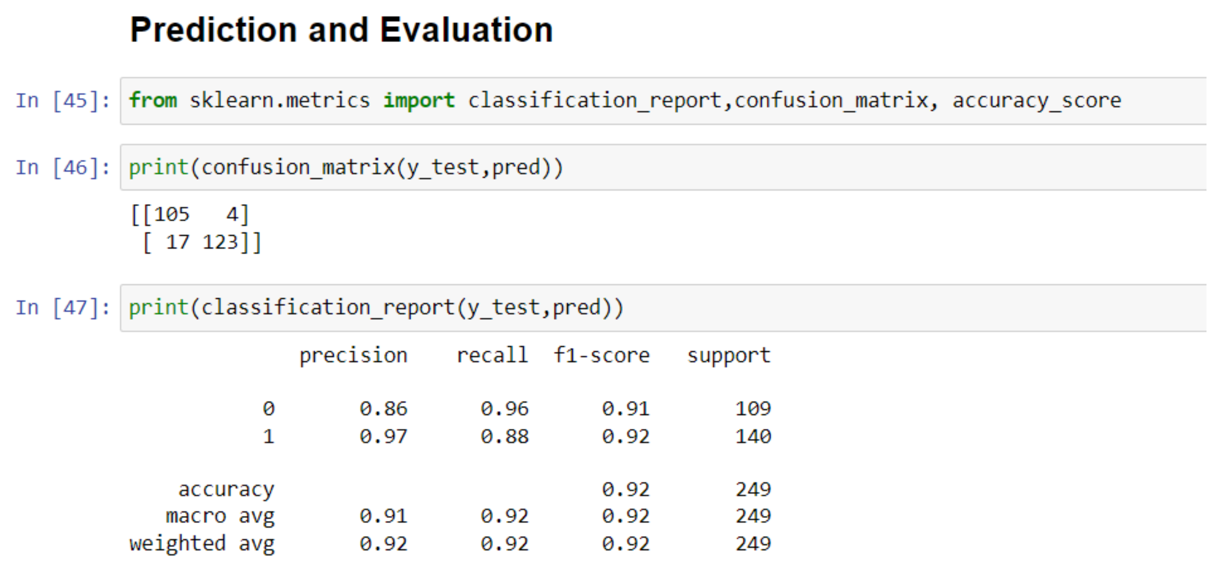
Evaluating the quality of the model is a fundamental part of the machine learning process. The most used performance evaluation metrics are calculated based on the elements of the confusion matrix.

- **Accuracy:** It represents the proportion of predictions that were correctly classified. Accuracy is the most commonly used evaluation metric; however, it is important to bear in mind that accuracy can be misleading when working with imbalanced datasets.

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$$

- **Precision:** It represents the proportion of positive predictions that are actually correct.

$$Precision = \frac{TP}{TP+FP}$$

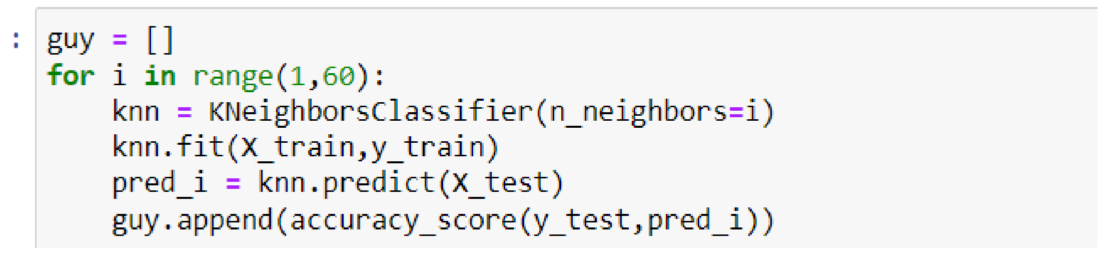


Prediction and Evaluation

```
In [45]: from sklearn.metrics import classification_report,confusion_matrix, accuracy_score

In [46]: print(confusion_matrix(y_test,pred))

[[105   4]
 [ 17 123]]

In [47]: print(classification_report(y_test,pred))

              precision    recall  f1-score   support

           0       0.86      0.96      0.91       109
           1       0.97      0.88      0.92       140

    accuracy                           0.92       249
   macro avg       0.91      0.92      0.92       249
weighted avg       0.92      0.92      0.92       249
```

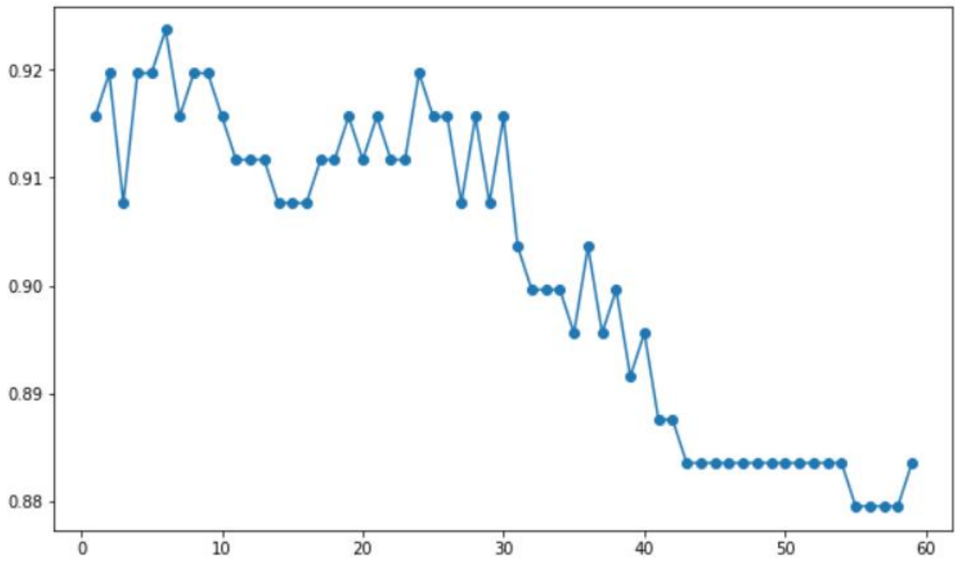
Drawing conclusions

After transforming the data, finally, we tuned the hyperparameters of the K-Nearest Neighbour (KNN) for model optimization, obtaining an accuracy of nearly 91% .

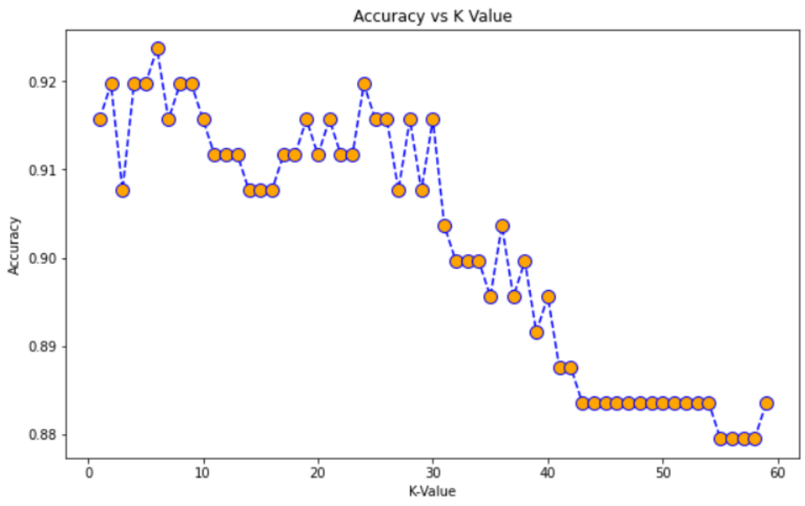


```
: guy = []
  for i in range(1,60):
      knn = KNeighborsClassifier(n_neighbors=i)
      knn.fit(X_train,y_train)
      pred_i = knn.predict(X_test)
      guy.append(accuracy_score(y_test,pred_i))
```

```
In [61]: plt.figure(figsize=(10,6))
         plt.scatter(range(1,60),guy)
         plt.plot(range(1,60),guy);
```



```
In [60]: plt.figure(figsize=(10,6))
         plt.plot(range(1,60),Accu,color='blue', linestyle='dashed', marker='o',markerfacecolor='orange', markersize=10)
         plt.title('Accuracy vs K Value')
         plt.xlabel('K-Value')
         plt.ylabel('Accuracy');
```



***

Predict whether the customer will churn(leave) the company or not.

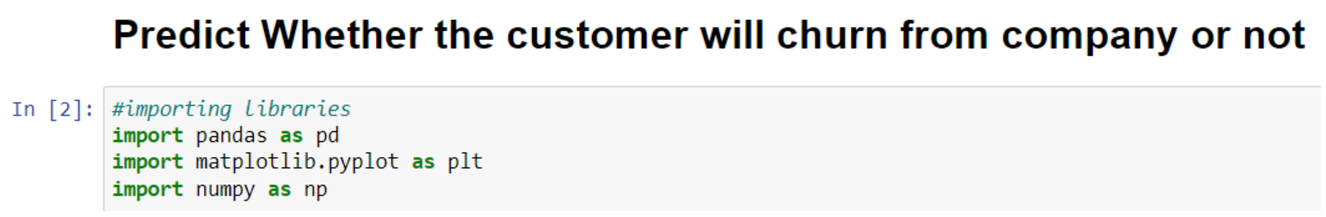
About the project

Predicting customer churn is critical for telecommunication companies to be able to effectively retain customers. It is more costly to acquire new customers than to retain existing ones. For this reason, large telecommunications corporations are seeking to develop models to predict which customers are more likely to change and take actions accordingly.

Dataset

The data set used in this project is provided by faculty in the google drive.
(Telco customer churn.csv)

Importing required libraries

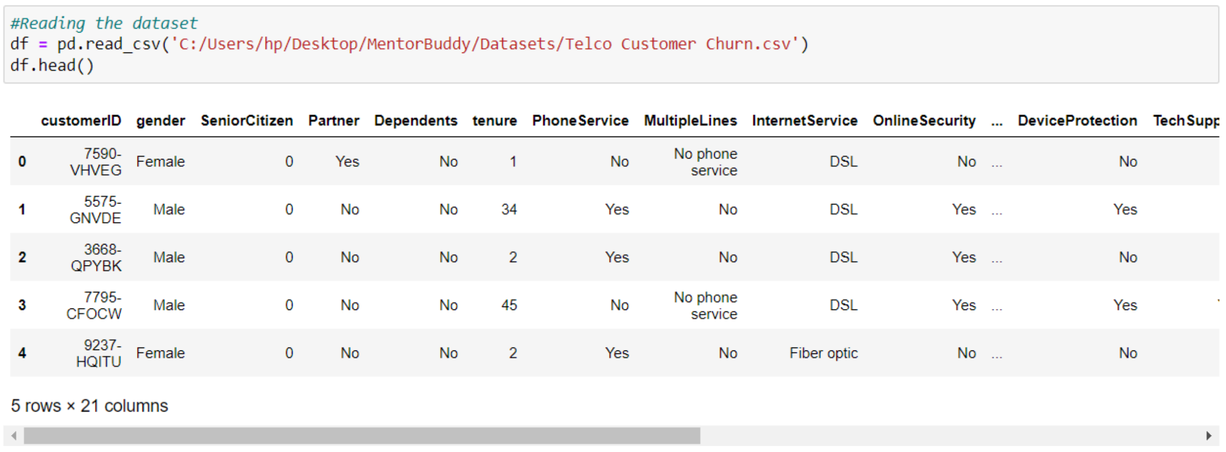


Predict Whether the customer will churn from company or not

```
In [2]: #importing libraries
        import pandas as pd
        import matplotlib.pyplot as plt
        import numpy as np
```

Reading Dataset

The first step of the analysis consists of reading and storing the data in a Pandas data frame using the pandas.read_csv function.

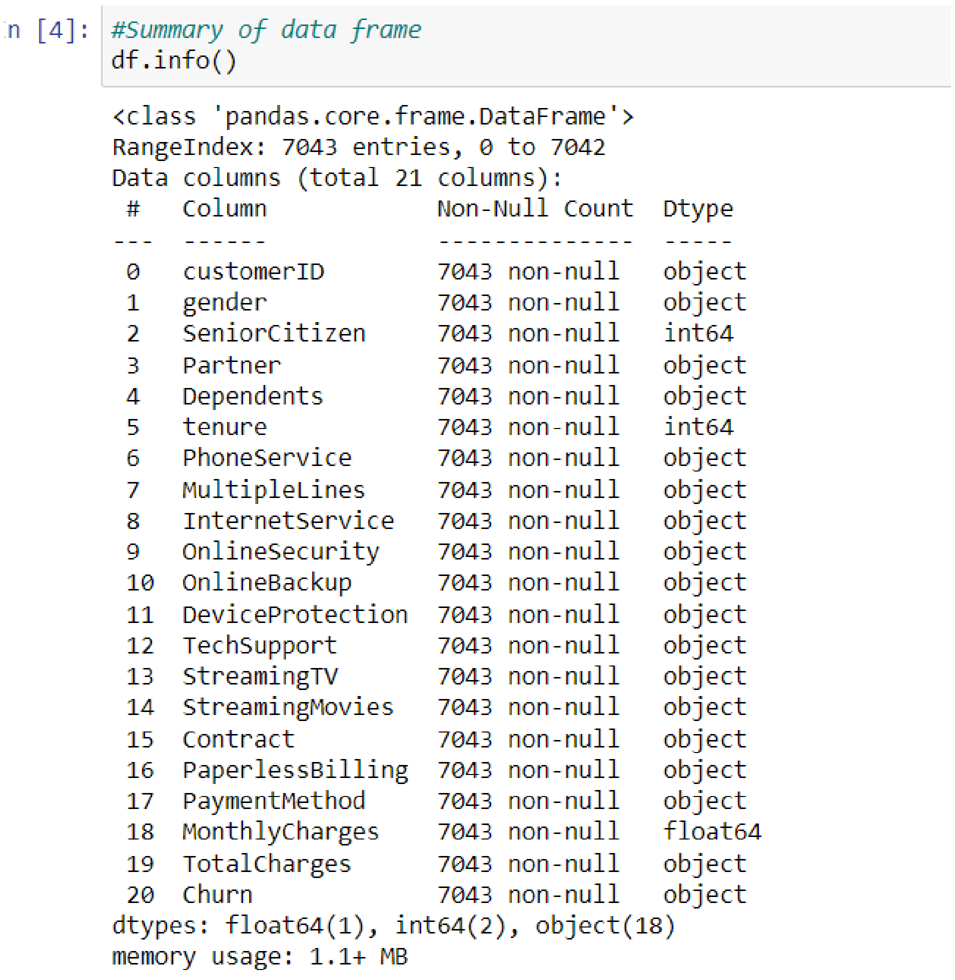


```
#Reading the dataset
df = pd.read_csv('C:/Users/hp/Desktop/MentorBuddy/Datasets/Telco Customer Churn.csv')
df.head()
```

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	...	DeviceProtection	TechSupp
0	7590-VHVEG	Female	0	Yes	No	1	No	No phone service	DSL	No	...	No	
1	5575-GNVDE	Male	0	No	No	34	Yes	No	DSL	Yes	...	Yes	
2	3668-QPYBK	Male	0	No	No	2	Yes	No	DSL	Yes	...	No	
3	7795-CFOCW	Male	0	No	No	45	No	No phone service	DSL	Yes	...	Yes	
4	9237-HQITU	Female	0	No	No	2	Yes	No	Fiber optic	No	...	No	

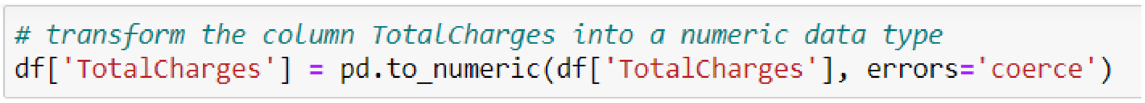
5 rows × 21 columns

As shown above, the data set contains 7043 observations and 21 columns. Apparently, there are no null values on the data set; however, we observe that the column TotalCharges was wrongly detected as an object. This column represents the total amount charged to the customer and it is, therefore, a numeric variable. For further analysis, we need to transform this column into a numeric data type. To do so, we can use the pd.to_numeric function. By default, this function raises an exception when it sees non-numeric data; however, we can use the argument errors='coerce' to skip those cases and replace them with a NaN.



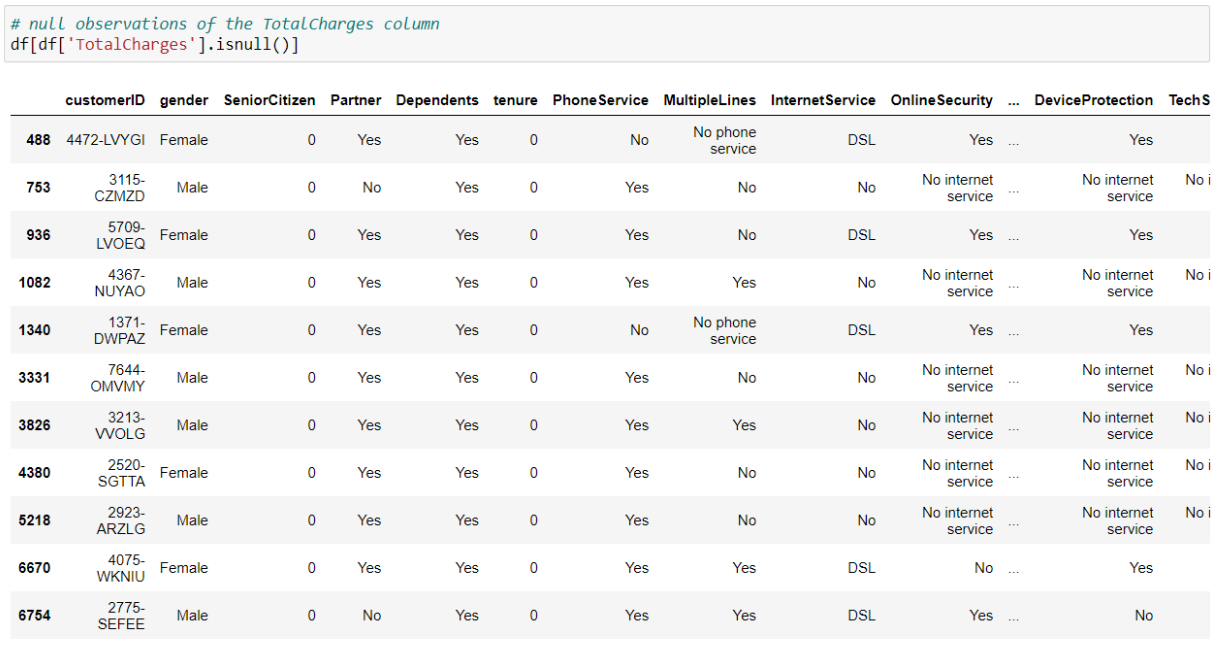
```
n [4]:  #Summary of data frame
        df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 21 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   customerID        7043 non-null   object 
 1   gender            7043 non-null   object 
 2   SeniorCitizen     7043 non-null   int64  
 3   Partner           7043 non-null   object 
 4   Dependents        7043 non-null   object 
 5   tenure            7043 non-null   int64  
 6   PhoneService      7043 non-null   object 
 7   MultipleLines     7043 non-null   object 
 8   InternetService   7043 non-null   object 
 9   OnlineSecurity    7043 non-null   object 
 10  OnlineBackup      7043 non-null   object 
 11  DeviceProtection  7043 non-null   object 
 12  TechSupport       7043 non-null   object 
 13  StreamingTV       7043 non-null   object 
 14  StreamingMovies   7043 non-null   object 
 15  Contract          7043 non-null   object 
 16  PaperlessBilling  7043 non-null   object 
 17  PaymentMethod     7043 non-null   object 
 18  MonthlyCharges    7043 non-null   float64
 19  TotalCharges      7043 non-null   object 
 20  Churn             7043 non-null   object 
dtypes: float64(1), int64(2), object(18)
memory usage: 1.1+ MB
```



```
# transform the column TotalCharges into a numeric data type
df['TotalCharges'] = pd.to_numeric(df['TotalCharges'], errors='coerce')
```

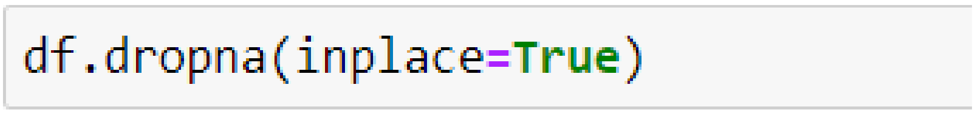
We can now observe that the column TotalCharges has 11 missing values



```
# null observations of the TotalCharges column
df[df['TotalCharges'].isnull()]
```

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	...	DeviceProtection	TechS
488	4472-LVYGI	Female	0	Yes	Yes	0	No	No phone service	DSL	Yes	...	Yes	
753	3115-CZMZD	Male	0	No	Yes	0	Yes	No	No	No internet service	...	No internet service	No i
936	5709-LVOEQ	Female	0	Yes	Yes	0	Yes	No	DSL	Yes	...	Yes	
1082	4367-NUYAO	Male	0	Yes	Yes	0	Yes	Yes	No	No internet service	...	No internet service	No i
1340	1371-DWPAZ	Female	0	Yes	Yes	0	No	No phone service	DSL	Yes	...	Yes	
3331	7644-OMVMY	Male	0	Yes	Yes	0	Yes	No	No	No internet service	...	No internet service	No i
3826	3213-VVOLG	Male	0	Yes	Yes	0	Yes	Yes	No	No internet service	...	No internet service	No i
4380	2520-SGTTA	Female	0	Yes	Yes	0	Yes	No	No	No internet service	...	No internet service	No i
5218	2923-ARZLG	Male	0	Yes	Yes	0	Yes	No	No	No internet service	...	No internet service	No i
6670	4075-WKNIU	Female	0	Yes	Yes	0	Yes	Yes	DSL	No	...	Yes	
6754	2775-SEFEE	Male	0	No	Yes	0	Yes	Yes	DSL	Yes	...	No	

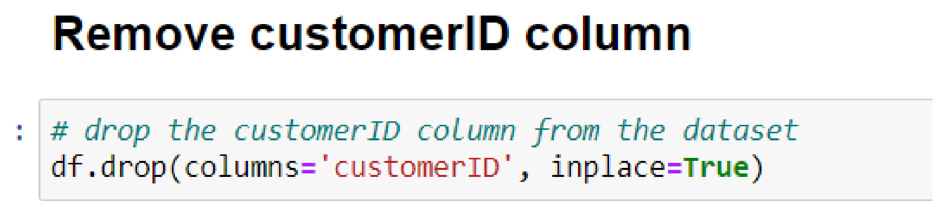
These observations have also a tenure of 0, even though MontlyCharges is not null for these entries. This information appeared to be contradictory, and therefore, we decide to remove those observations from the data set.



```
df.dropna(inplace=True)
```

Remove customerID column

The customerID column is useless to explain whether not the customer will churn. Therefore, we drop this column from the data set.

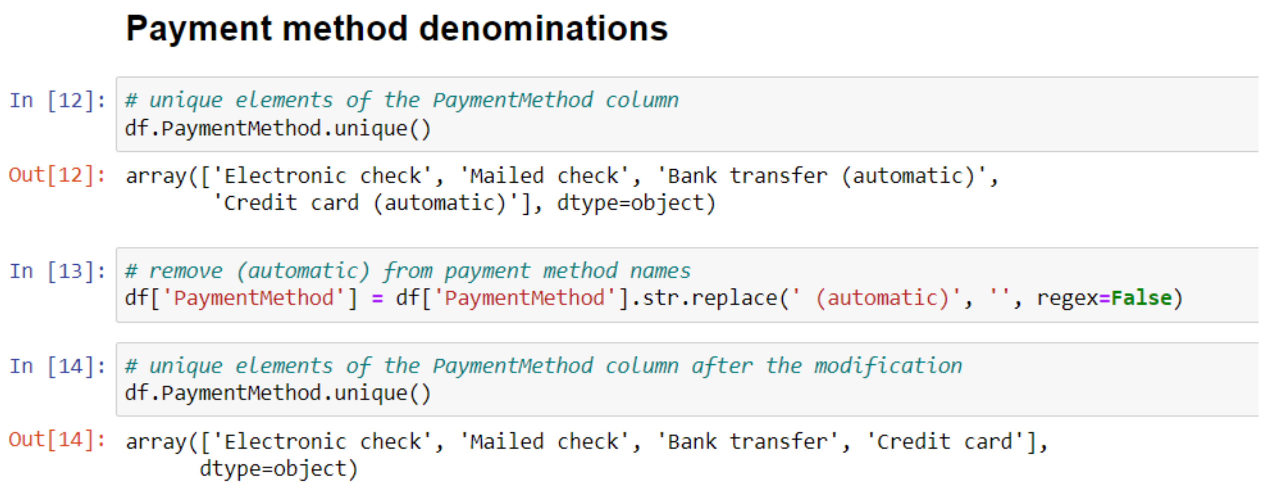


Remove customerID column

```
: # drop the customerID column from the dataset
  df.drop(columns='customerID', inplace=True)
```

Payment method denominations

As shown below, some payment method denominations contain in parenthesis the word automatic. These denominations are too long to be used as tick labels in further visualizations. Therefore, we remove this clarification in parenthesis from the entries of the PaymentMethod column.



Payment method denominations

```
In [12]: # unique elements of the PaymentMethod column
         df.PaymentMethod.unique()

Out[12]: array(['Electronic check', 'Mailed check', 'Bank transfer (automatic)',
                'Credit card (automatic)'], dtype=object)

In [13]: # remove (automatic) from payment method names
         df['PaymentMethod'] = df['PaymentMethod'].str.replace(' (automatic)', '', regex=False)

In [14]: # unique elements of the PaymentMethod column after the modification
         df.PaymentMethod.unique()

Out[14]: array(['Electronic check', 'Mailed check', 'Bank transfer', 'Credit card'],
               dtype=object)
```

Data Visualization

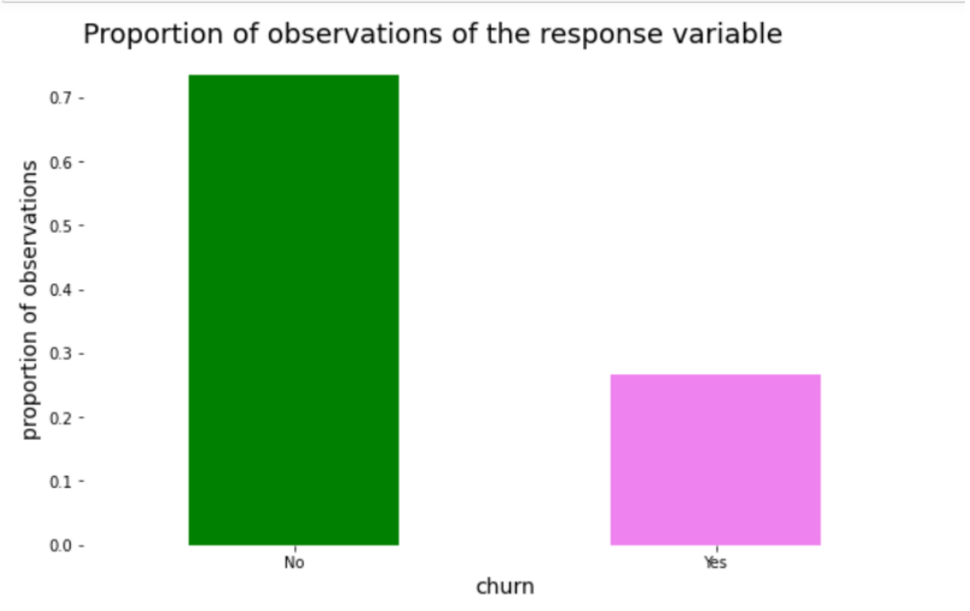
Response Variable

The following bar plot shows the percentage of observations that correspond to each class of the response variable: no and yes. As shown below, this is an imbalanced data set because both classes are not equally distributed among all observations, being no the majority class (73.42%). When modeling, this imbalance will lead to a large number of false negatives, as we will see later.

Data Visualization

Response Variable

```
[6]: # create a figure
fig = plt.figure(figsize=(10, 6))
ax = fig.add_subplot(111)

# proportion of observation of each class
prop_response = df['Churn'].value_counts(normalize=True)

# create a bar plot showing the percentage of churn
prop_response.plot(kind='bar',
                   ax=ax,
                   color=['green','violet'])

# set title and labels
ax.set_title('Proportion of observations of the response variable',
             fontsize=18, loc='left')
ax.set_xlabel('churn',
              fontsize=14)
ax.set_ylabel('proportion of observations',
              fontsize=14)
ax.tick_params(rotation='auto')

# eliminate the frame from the plot
spine_names = ('top', 'right', 'bottom', 'left')
for spine_name in spine_names:
    ax.spines[spine_name].set_visible(False)
```



A normalized stacked bar plot makes each column the same height, so it is not useful for comparing total numbers; however, it is perfect for comparing how the response variable varies across all groups of an independent variable.

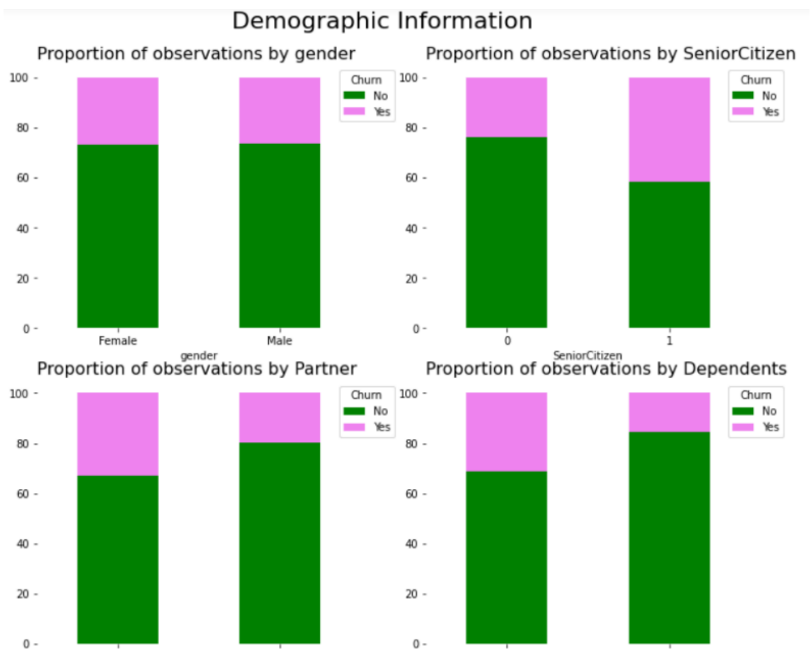
Demographic Information

The following code creates a stacked percentage bar chart for each demographic attribute (gender, SeniorCitizen, Partner, Dependents), showing the percentage of Churn for each category of the attribute.

```python
def percentage_stacked_plot(columns_to_plot, super_title):

    number_of_columns = 2
    number_of_rows = math.ceil(len(columns_to_plot)/2)

    # create a figure
    fig = plt.figure(figsize=(12, 5 * number_of_rows))
    fig.suptitle(super_title, fontsize=22,  y=.95)


    # loop to each column name to create a subplot
    for index, column in enumerate(columns_to_plot, 1):

        # create the subplot
        ax = fig.add_subplot(number_of_rows, number_of_columns, index)

        # calculate the percentage of observations of the response variable for each group of the independent variable
        # 100% stacked bar plot
        prop_by_independent = pd.crosstab(df[column], df['Churn']).apply(lambda x: x/x.sum()*100, axis=1)

        prop_by_independent.plot(kind='bar', ax=ax, stacked=True,
                                 rot=0, color=['green','violet'])

        # set the legend in the upper right corner
        ax.legend(loc="upper right", bbox_to_anchor=(0.62, 0.5, 0.5, 0.5),
                  title='Churn', fancybox=True)

        # set title and labels
        ax.set_title('Proportion of observations by ' + column,
                     fontsize=16, loc='left')

        ax.tick_params(rotation='auto')

        # eliminate the frame from the plot
        spine_names = ('top', 'right', 'bottom', 'left')
        for spine_name in spine_names:
            ax.spines[spine_name].set_visible(False)
```

]:

```python
import math
# demographic column names
demographic_columns = ['gender', 'SeniorCitizen', 'Partner', 'Dependents']

# stacked plot of demographic columns
percentage_stacked_plot(demographic_columns, 'Demographic Information')
```



As shown above, each bar is a category of the independent variable, and it is subdivided to show the proportion of each response class (No and Yes).

We can extract the following conclusions by analyzing demographic attributes:

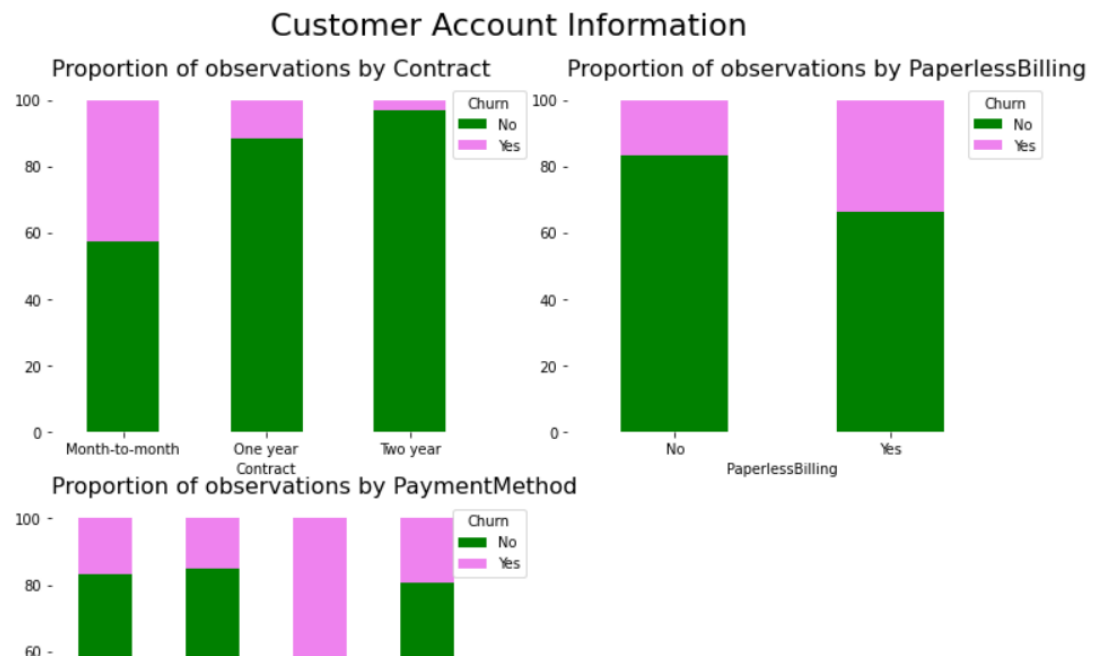
- The churn rate of senior citizens is almost double that of young citizens.
- We do not expect gender to have significant predictive power. A similar percentage of churn is shown both when a customer is a man or a woman.
- Customers with a partner churn less than customers with no partner.

Customer Account Information — Categorical variables

As we did with demographic attributes, we evaluate the percentage of Churn for each category of the customer account attributes (Contract, PaperlessBilling, PaymentMethod).

Categorical variables

```
# customer account column names
account_columns = ['Contract', 'PaperlessBilling', 'PaymentMethod']

# stacked plot of customer account columns
percentage_stacked_plot(account_columns, 'Customer Account Information')
```

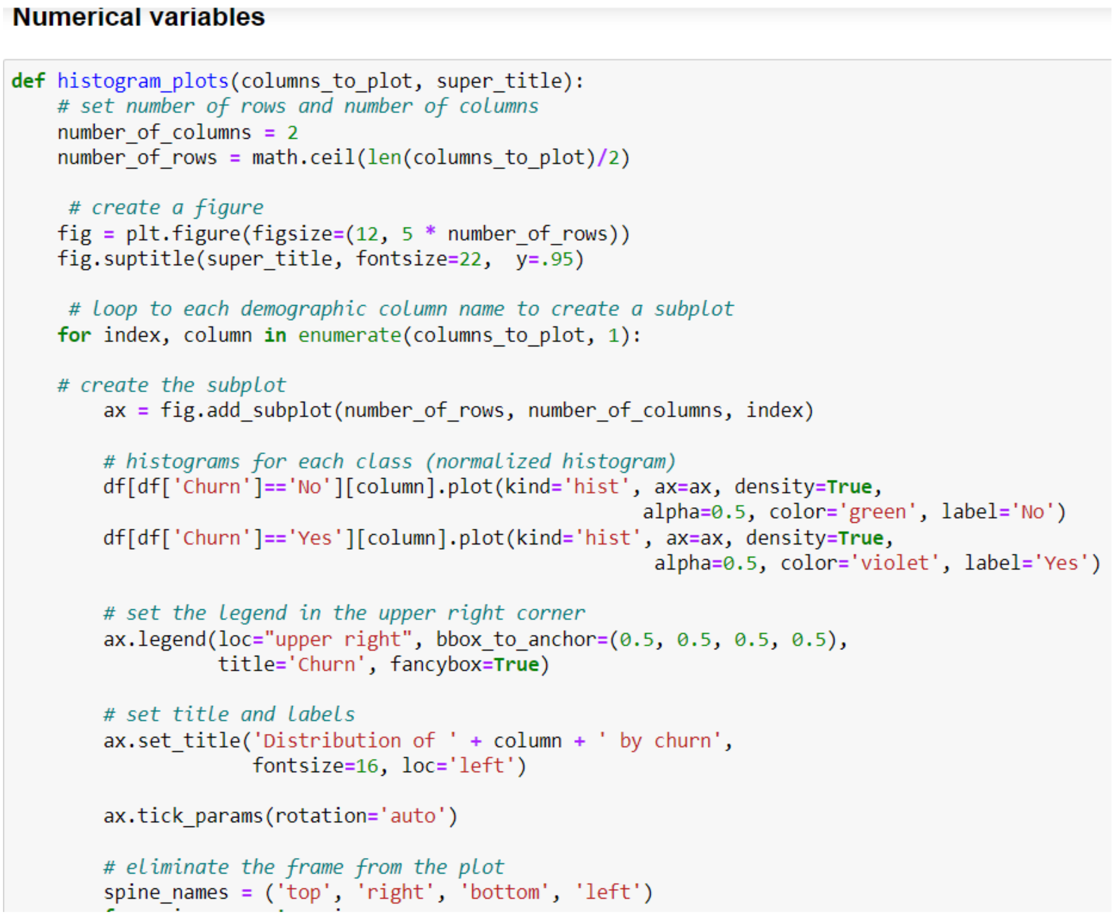


We can extract the following conclusions by analyzing customer account attributes:

- Customers with month-to-month contracts have higher churn rates compared to clients with yearly contracts.
- Customers who opted for an electronic check as paying method are more likely to leave the company.
- Customers subscribed to paperless billing churn more than those who are not subscribed.

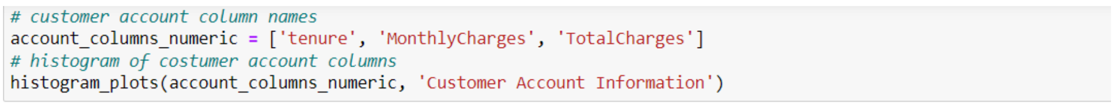
Customer Account Information — Numerical variables

The following plots show the distribution of tenure, MontlyCharges, TotalCharges by Churn. For all numeric attributes, the distributions of both classes (No and Yes) are different which suggests that all of the attributes will be useful to determine whether or not a customer churns.

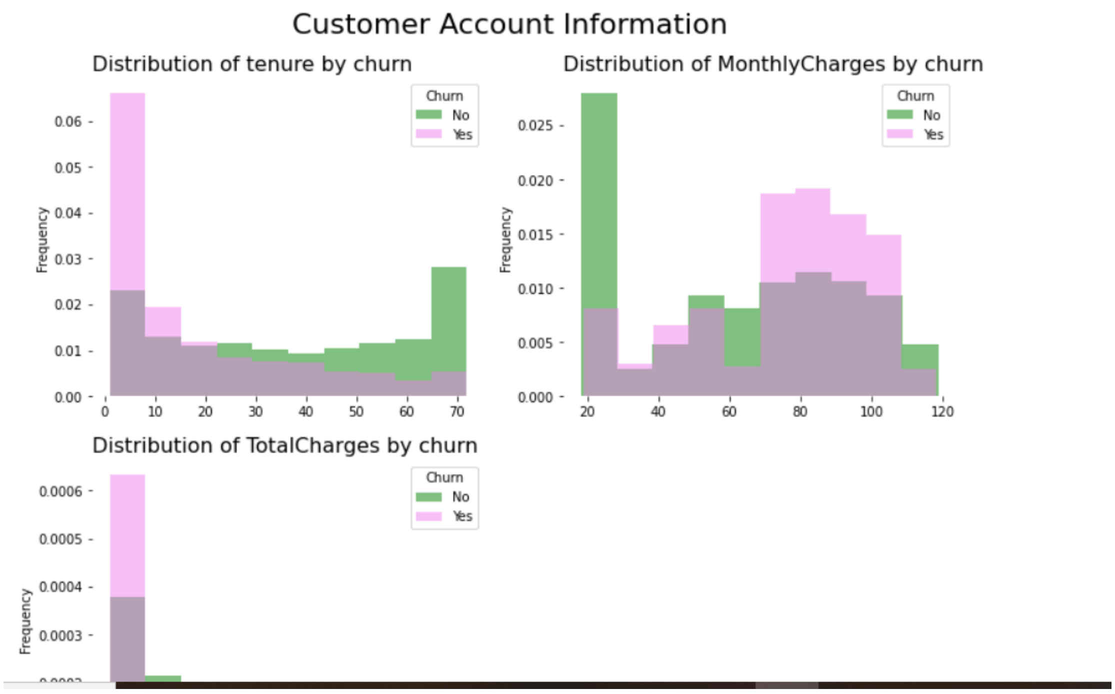


Numerical variables

```
def histogram_plots(columns_to_plot, super_title):
    # set number of rows and number of columns
    number_of_columns = 2
    number_of_rows = math.ceil(len(columns_to_plot)/2)

     # create a figure
    fig = plt.figure(figsize=(12, 5 * number_of_rows))
    fig.suptitle(super_title, fontsize=22,  y=.95)

     # loop to each demographic column name to create a subplot
    for index, column in enumerate(columns_to_plot, 1):

    # create the subplot
        ax = fig.add_subplot(number_of_rows, number_of_columns, index)

        # histograms for each class (normalized histogram)
        df[df['Churn']=='No'][column].plot(kind='hist', ax=ax, density=True,
                                               alpha=0.5, color='green', label='No')
        df[df['Churn']=='Yes'][column].plot(kind='hist', ax=ax, density=True,
                                                alpha=0.5, color='violet', label='Yes')

        # set the legend in the upper right corner
        ax.legend(loc="upper right", bbox_to_anchor=(0.5, 0.5, 0.5, 0.5),
                  title='Churn', fancybox=True)

        # set title and labels
        ax.set_title('Distribution of ' + column + ' by churn',
                     fontsize=16, loc='left')

        ax.tick_params(rotation='auto')

        # eliminate the frame from the plot
        spine_names = ('top', 'right', 'bottom', 'left')
```

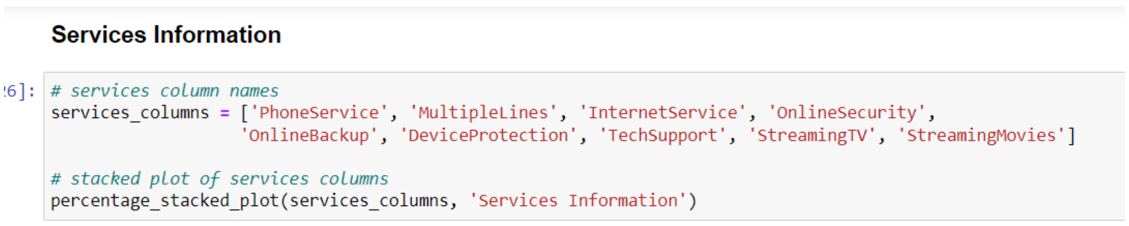


```
# customer account column names
account_columns_numeric = ['tenure', 'MonthlyCharges', 'TotalCharges']
# histogram of costumer account columns
histogram_plots(account_columns_numeric, 'Customer Account Information')
```



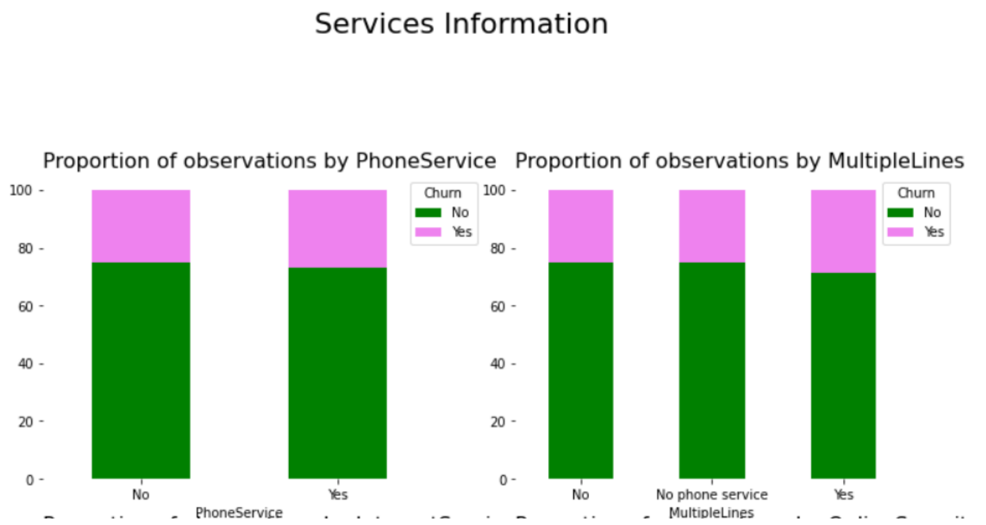
Services Information

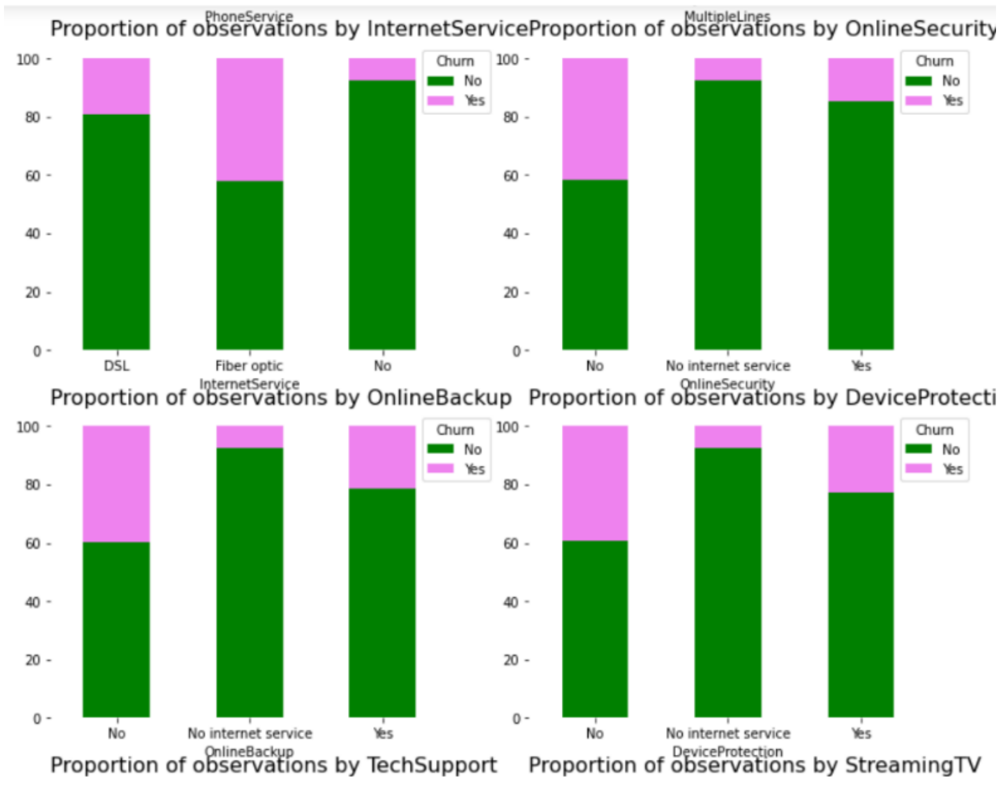
Lastly, we evaluate the percentage of the target for each category of the services columns with stacked bar plots.

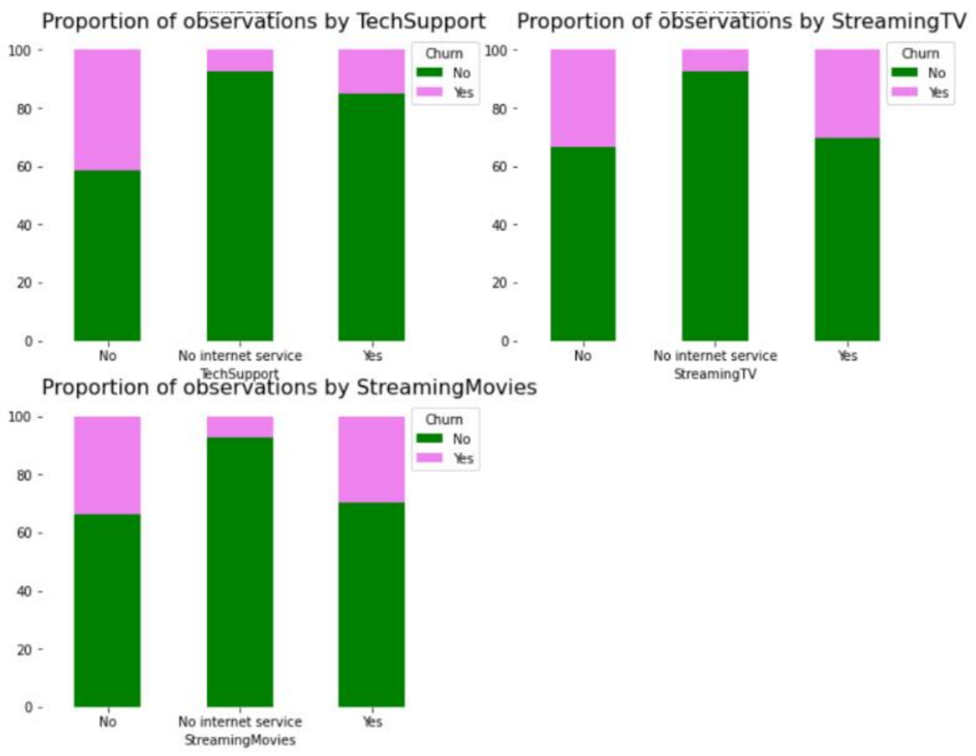


Services Information

```
:6]: # services column names
     services_columns = ['PhoneService', 'MultipleLines', 'InternetService', 'OnlineSecurity',
                         'OnlineBackup', 'DeviceProtection', 'TechSupport', 'StreamingTV', 'StreamingMovies']

     # stacked plot of services columns
     percentage_stacked_plot(services_columns, 'Services Information')
```





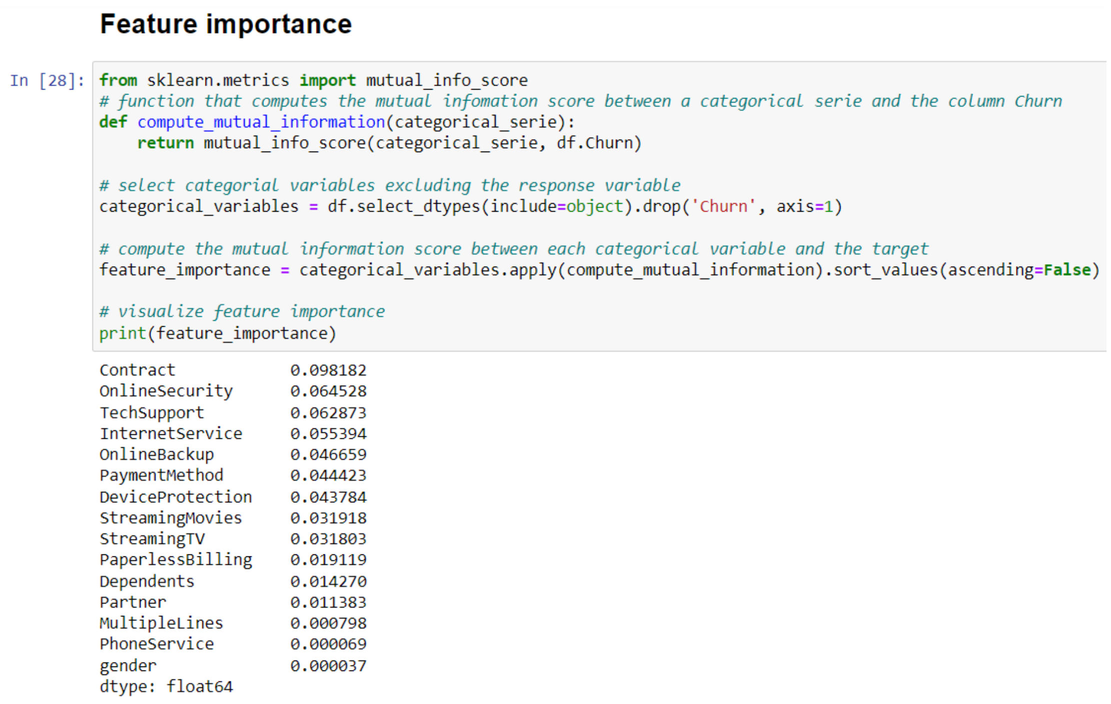


By looking at the plots above, we can identify the most relevant attributes for detecting churn. We expect these attributes to be discriminative in our future models.

Feature importance

Mutual information — analysis of linear and nonlinear relationships

Mutual information measures the mutual dependency between two variables based on entropy estimations. In machine learning, we are interested in evaluating the degree of dependency between each independent variable and the response variable. Higher values of mutual information show a higher degree of dependency which indicates that the independent variable will be useful for predicting the target.



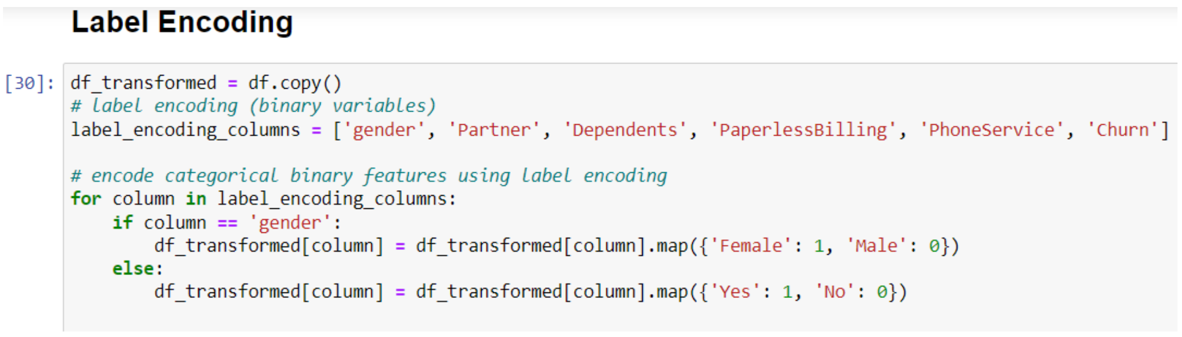
Feature importance

```
In [28]: from sklearn.metrics import mutual_info_score
         # function that computes the mutual infomation score between a categorical serie and the column Churn
         def compute_mutual_information(categorical_serie):
             return mutual_info_score(categorical_serie, df.Churn)

         # select categorial variables excluding the response variable
         categorical_variables = df.select_dtypes(include=object).drop('Churn', axis=1)

         # compute the mutual information score between each categorical variable and the target
         feature_importance = categorical_variables.apply(compute_mutual_information).sort_values(ascending=False)

         # visualize feature importance
         print(feature_importance)
```

```
Contract            0.098182
OnlineSecurity      0.064528
TechSupport         0.062873
InternetService     0.055394
OnlineBackup        0.046659
PaymentMethod       0.044423
DeviceProtection    0.043784
StreamingMovies     0.031918
StreamingTV         0.031803
PaperlessBilling    0.019119
Dependents          0.014270
Partner             0.011383
MultipleLines       0.000798
PhoneService        0.000069
gender              0.000037
dtype: float64
```

Label Encoding

Label encoding is used to replace categorical values with numerical values. This encoding replaces every category with a numerical label. In this project, we use label encoding with the following binary variables: (1) gender, (2) Partner, (3) Dependents, (4)PaperlessBilling, (5)PhoneService , and (6)Churn .

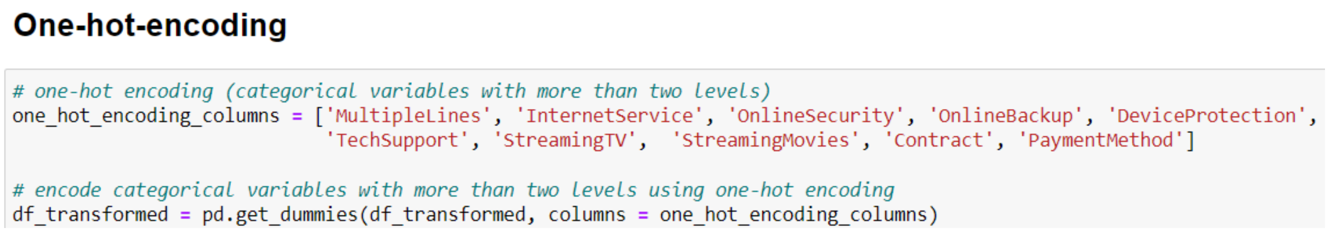


Label Encoding

```
[30]: df_transformed = df.copy()
      # label encoding (binary variables)
      label_encoding_columns = ['gender', 'Partner', 'Dependents', 'PaperlessBilling', 'PhoneService', 'Churn']

      # encode categorical binary features using label encoding
      for column in label_encoding_columns:
          if column == 'gender':
              df_transformed[column] = df_transformed[column].map({'Female': 1, 'Male': 0})
          else:
              df_transformed[column] = df_transformed[column].map({'Yes': 1, 'No': 0})
```

One-Hot Encoding

One-hot encoding creates a new binary column for each level of the categorical variable. The new column contains zeros and ones indicating the absence or presence of the category in the data. In this project, we apply one-hot encoding to the following categorical variables: (1) Contract, (2) PaymentMethod, (3) MultipleLines, (4) InternetServices, (5) OnlineSecurity, (6) OnlineBackup, (7) DeviceProtection, (8) TechSupport, (9) StreamingTV, and (10)StreamingMovies.

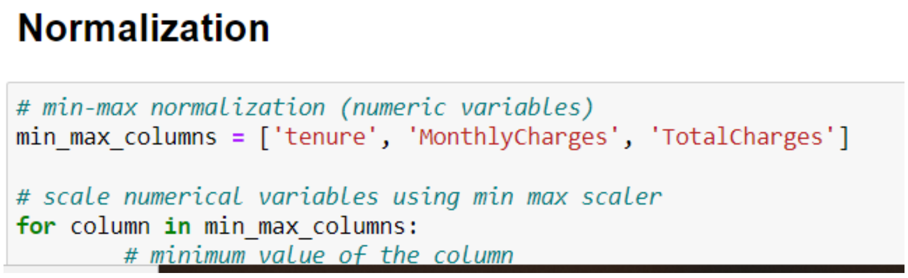


One-hot-encoding

```
# one-hot encoding (categorical variables with more than two levels)
one_hot_encoding_columns = ['MultipleLines', 'InternetService', 'OnlineSecurity', 'OnlineBackup', 'DeviceProtection',
                            'TechSupport', 'StreamingTV',  'StreamingMovies', 'Contract', 'PaymentMethod']

# encode categorical variables with more than two levels using one-hot encoding
df_transformed = pd.get_dummies(df_transformed, columns = one_hot_encoding_columns)
```

Normalization

Data Normalization is a common practice in machine learning which consists of transforming numeric columns to a common scale. In machine learning, some feature values differ from others multiple times. The features with higher values will dominate the learning process; however, it does not mean those variables are more important to predict the target. Data normalization transforms multiscaled data to the same scale. After normalization, all variables have a similar influence on the model, improving the stability and performance of the learning algorithm.



Normalization

```
# min-max normalization (numeric variables)
min_max_columns = ['tenure', 'MonthlyCharges', 'TotalCharges']

# scale numerical variables using min max scaler
for column in min_max_columns:
        # minimum value of the column
```

```
        min_column = df_transformed[column].min()
        # maximum value of the column
        max_column = df_transformed[column].max()
```

```
# min max scaler
df_transformed[column] = (df_transformed[column] - min_column) / (max_column - min_column)
```

Splitting the data in training and testing sets

The first step when building a model is to split the data into two groups, which are typically referred to as training and testing sets.

```
# select independent variables
X = df_transformed.drop(columns='Churn')

# select dependent variables
y = df_transformed.loc[:, 'Churn']

# prove that the variables were selected correctly
print(X.columns)

# prove that the variables were selected correctly
print(y.name)
```

```
Index(['gender', 'SeniorCitizen', 'Partner', 'Dependents', 'tenure',
       'PhoneService', 'PaperlessBilling', 'MonthlyCharges', 'TotalCharges',
       'MultipleLines_No', 'MultipleLines_No phone service',
       'MultipleLines_Yes', 'InternetService_DSL',
       'InternetService_Fiber optic', 'InternetService_No',
       'OnlineSecurity_No', 'OnlineSecurity_No internet service',
       'OnlineSecurity_Yes', 'OnlineBackup_No',
       'OnlineBackup_No internet service', 'OnlineBackup_Yes',
       'DeviceProtection_No', 'DeviceProtection_No internet service',
       'DeviceProtection_Yes', 'TechSupport_No',
       'TechSupport_No internet service', 'TechSupport_Yes', 'StreamingTV_No',
       'StreamingTV_No internet service', 'StreamingTV_Yes',
       'StreamingMovies_No', 'StreamingMovies_No internet service',
       'StreamingMovies_Yes', 'Contract_Month-to-month', 'Contract_One year',
       'Contract_Two year', 'PaymentMethod_Bank transfer',
       'PaymentMethod_Credit card', 'PaymentMethod_Electronic check',
       'PaymentMethod_Mailed check'],
      dtype='object')
Churn
```

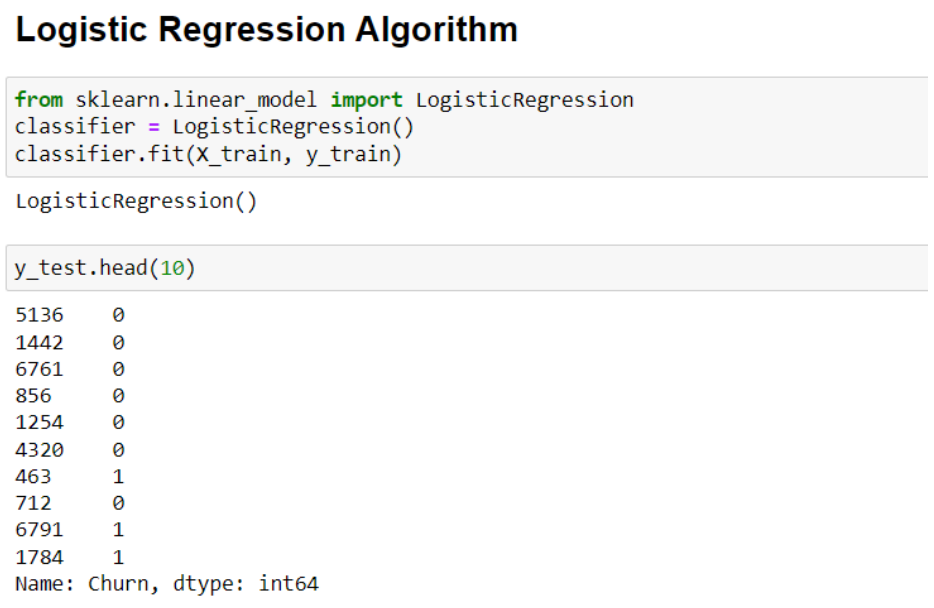
Splitting the data in training and testing sets

```
# split the data in training and testing sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,random_state=51, shuffle=True)
```

Assessing algorithm

Algorithm selection is a key challenge in any machine learning project since there is not an algorithm that is the best across all projects. Generally, we need to evaluate a set of potential candidates and select for further evaluation those that provide better performance.

In this project, I’m using Logistic Regression algorithms to implement in Scikit-Learn.


Logistic Regression Algorithm

```
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression()
classifier.fit(X_train, y_train)
```

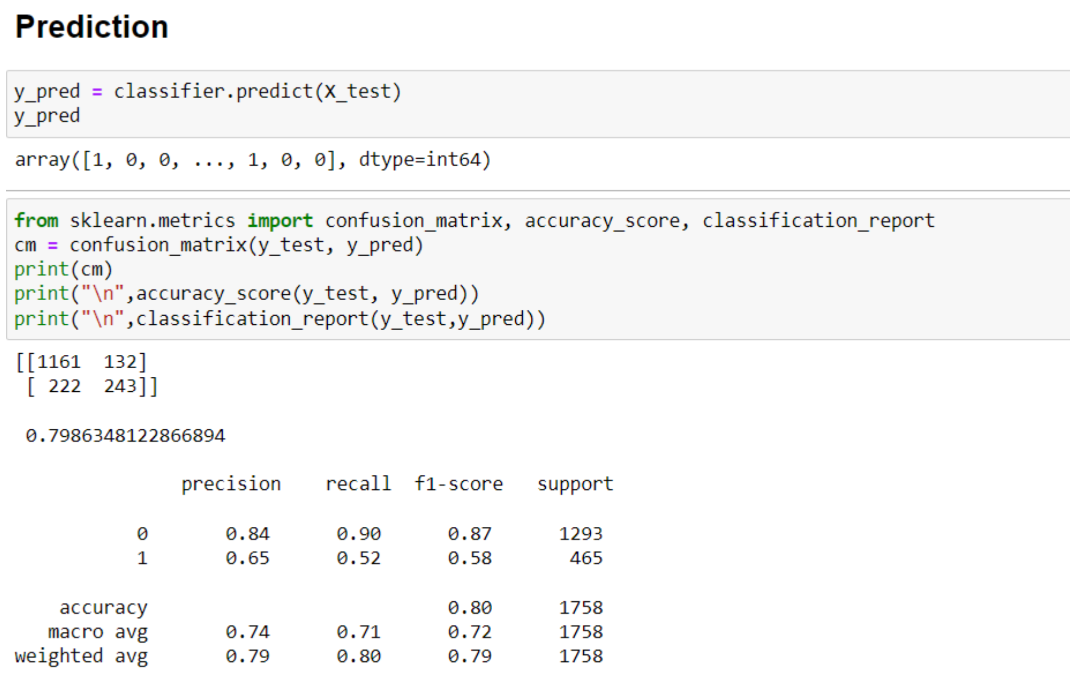
```
LogisticRegression()
```

```
y_test.head(10)
```

```
5136    0
1442    0
6761    0
856     0
1254    0
4320    0
463     1
712     0
6791    1
1784    1
Name: Churn, dtype: int64
```

Evaluation metrics

Evaluating the quality of the model is a fundamental part of the machine learning process. The most used performance evaluation metrics are calculated based on the elements of the confusion matrix.


Prediction

```
y_pred = classifier.predict(X_test)
y_pred
```

```
array([1, 0, 0, ..., 1, 0, 0], dtype=int64)
```

```
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
cm = confusion_matrix(y_test, y_pred)
print(cm)
print("\n",accuracy_score(y_test, y_pred))
print("\n",classification_report(y_test,y_pred))
```

```
[[1161  132]
 [ 222  243]]

 0.7986348122866894

              precision    recall  f1-score   support

           0       0.84      0.90      0.87      1293
           1       0.65      0.52      0.58       465

    accuracy                           0.80      1758
   macro avg       0.74      0.71      0.72      1758
weighted avg       0.79      0.80      0.79      1758
```

Conclusion

In this project, I have walked through a complete end-to-end machine learning project using the Telco customer Churn dataset. I started by cleaning the data and analyzing it with visualization. Then, to be able to build a machine learning model, we transformed the categorical data into numeric variables (feature engineering). After transforming the data, I tried Logistic Regression for model optimization, obtaining an accuracy of nearly 80%.

Prediction of House prices

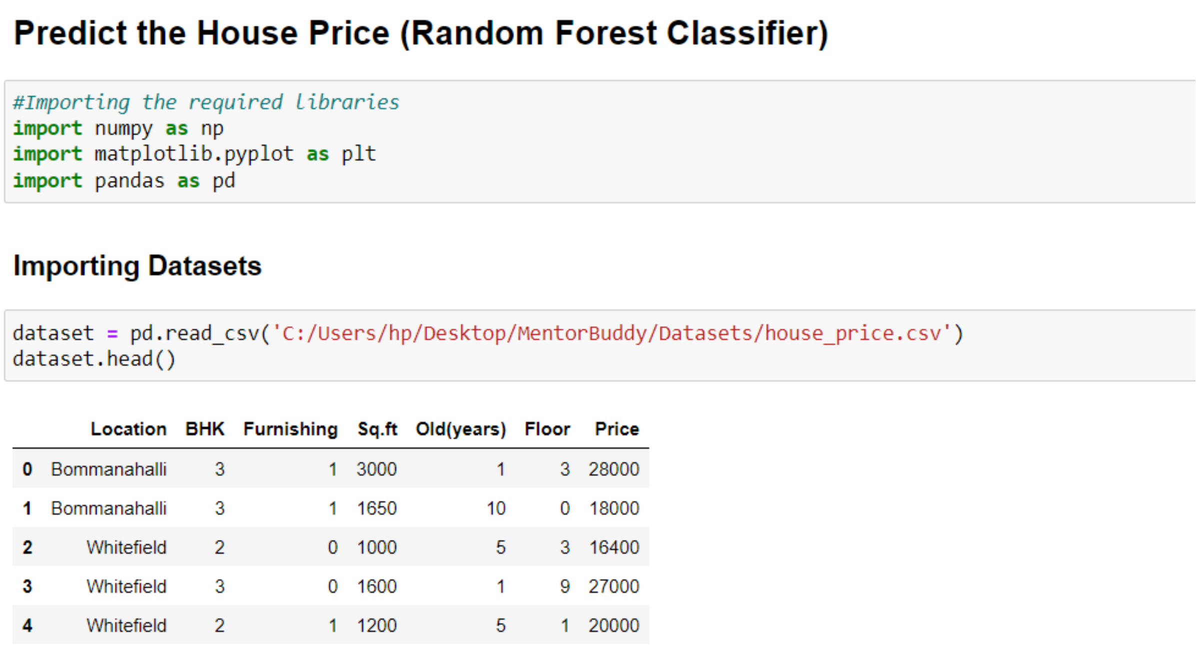
About the project

My goal for this project is to build an end to end solution or application that is capable of predicting the house prices better than individuals. Another motivation for this project is to implement similar solution at my workplace and help Investments and Residential team make data driven decisions.

Understanding Data

Dataset used : House price

Importing Libraries and dataset



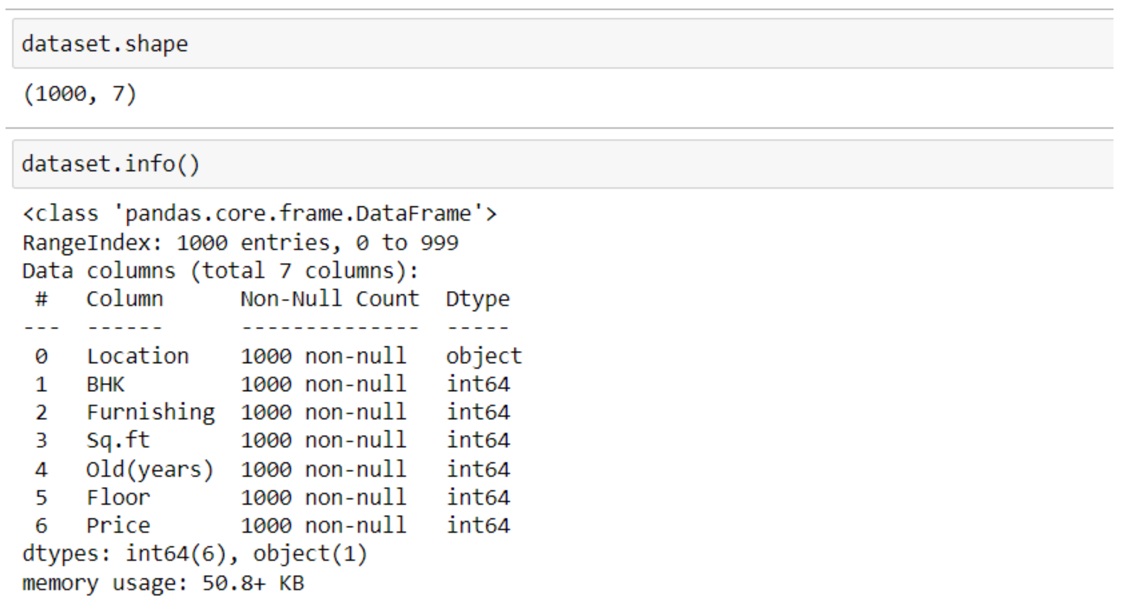
Predict the House Price (Random Forest Classifier)

```
#Importing the required libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

Importing Datasets

```
dataset = pd.read_csv('C:/Users/hp/Desktop/MentorBuddy/Datasets/house_price.csv')
dataset.head()
```

	Location	BHK	Furnishing	Sq.ft	Old(years)	Floor	Price
0	Bommanahalli	3	1	3000	1	3	28000
1	Bommanahalli	3	1	1650	10	0	18000
2	Whitefield	2	0	1000	5	3	16400
3	Whitefield	3	0	1600	1	9	27000
4	Whitefield	2	1	1200	5	1	20000



```
dataset.shape
```

```
(1000, 7)
```

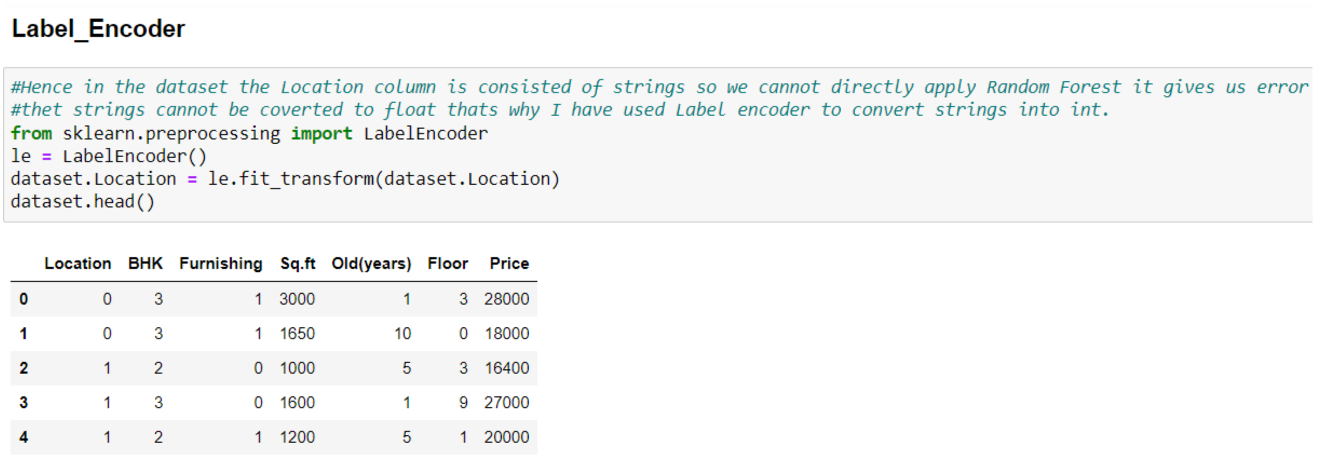
```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 7 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   Location    1000 non-null   object
 1   BHK         1000 non-null   int64 
 2   Furnishing  1000 non-null   int64 
 3   Sq.ft       1000 non-null   int64 
 4   Old(years)  1000 non-null   int64 
 5   Floor       1000 non-null   int64 
 6   Price       1000 non-null   int64 
dtypes: int64(6), object(1)
memory usage: 50.8+ KB
```

There are 1000 observations in the data set with zero missing values. Also, we can see that all the features are numeric except Location which is categorical variable.

Label Encoder

Since the Location column contains the categorical data we have to convert it into numerical data using Label encoder

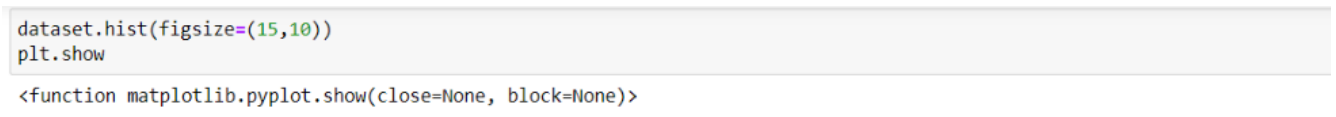


Label_Encoder

```
#Hence in the dataset the Location column is consisted of strings so we cannot directly apply Random Forest it gives us error
#thet strings cannot be coverted to float thats why I have used Label encoder to convert strings into int.
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
dataset.Location = le.fit_transform(dataset.Location)
dataset.head()
```

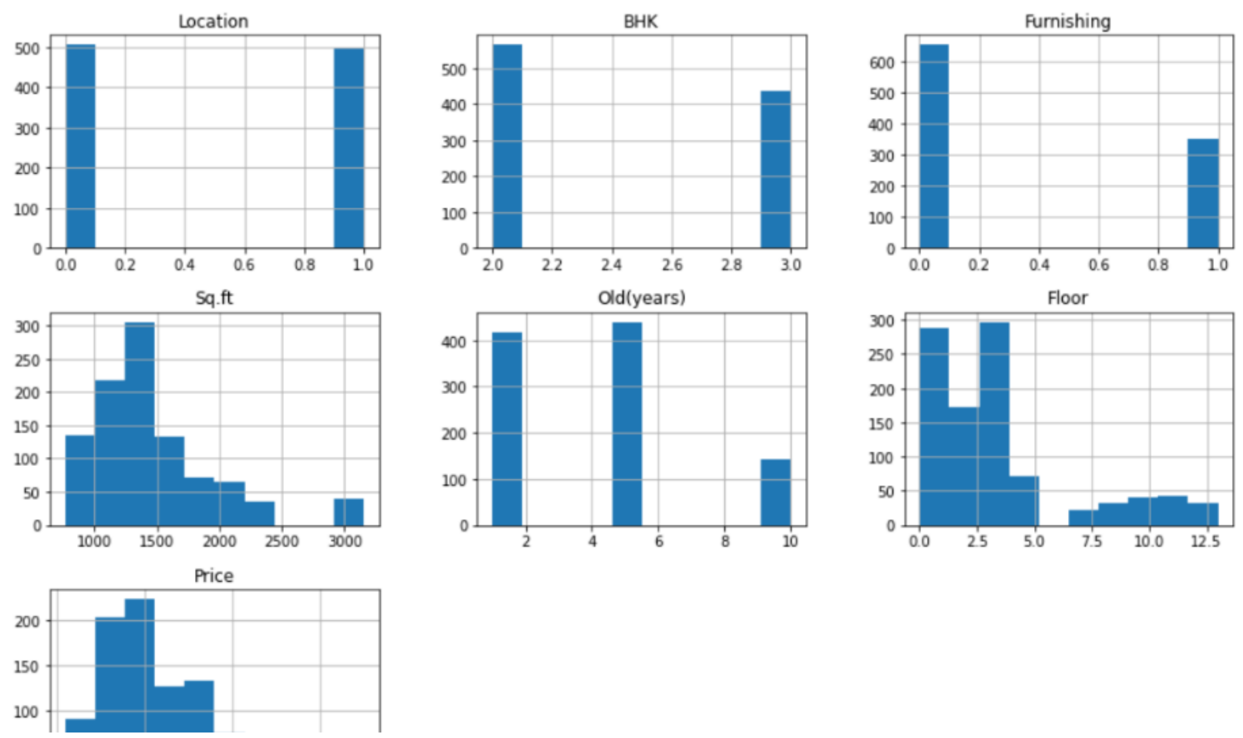
	Location	BHK	Furnishing	Sq.ft	Old(years)	Floor	Price
0	0	3	1	3000	1	3	28000
1	0	3	1	1650	10	0	18000
2	1	2	0	1000	5	3	16400
3	1	3	0	1600	1	9	27000
4	1	2	1	1200	5	1	20000

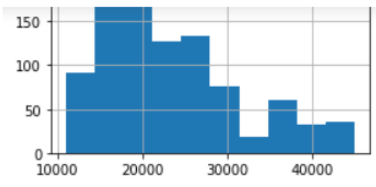
Data Visualization



```
dataset.hist(figsize=(15,10))
plt.show
```

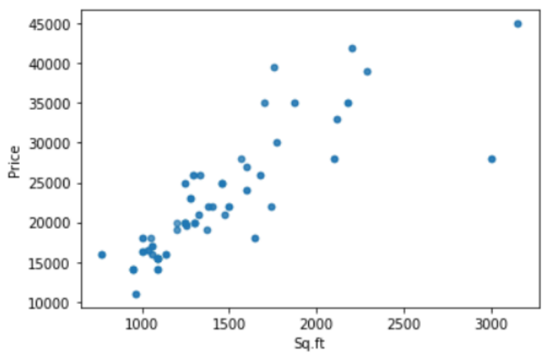
```
<function matplotlib.pyplot.show(close=None, block=None)>
```





```
dataset.plot(kind="scatter", x="Sq.ft",y="Price", alpha=0.1)
```

```
<AxesSubplot:xlabel='Sq.ft', ylabel='Price'>
```



Here we can see how price of house varies with square ft. and also the histogram represents the data visualization of the different features.

```
x = dataset.iloc[:, : 6]
y = dataset.iloc[:, 6]
```

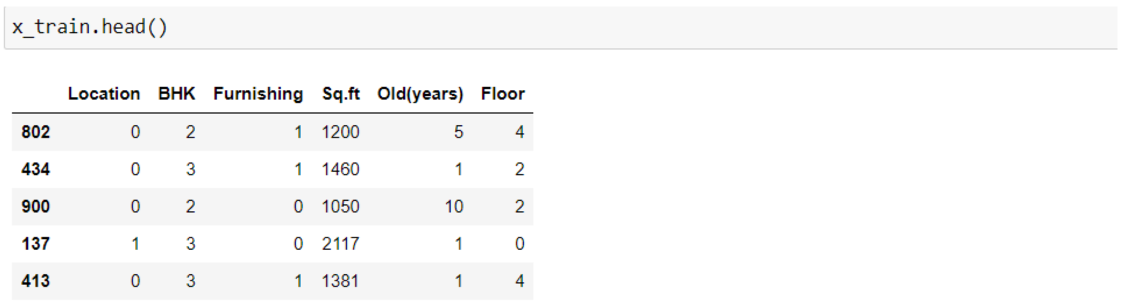
```
x
y
```

```
0       28000
1       18000
2       16400
3       27000
4       20000
        ...
995     25000
996     28000
```

Splitting the data for Training and Testing

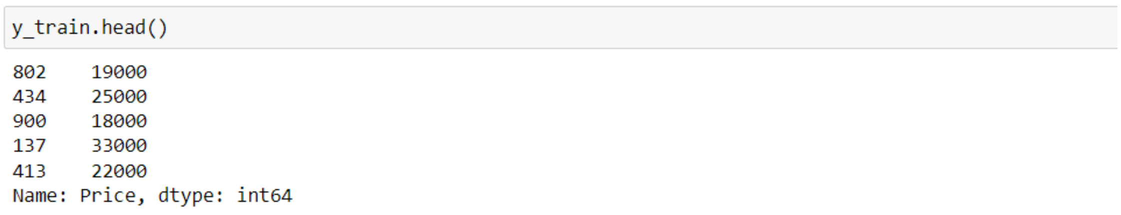
Splitting the dataset into training-set and testing-set

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split (x, y, test_size = 0.3, random_state = 30)
```



```
x_train.head()
```

	Location	BHK	Furnishing	Sq.ft	Old(years)	Floor
802	0	2	1	1200	5	4
434	0	3	1	1460	1	2
900	0	2	0	1050	10	2
137	1	3	0	2117	1	0
413	0	3	1	1381	1	4



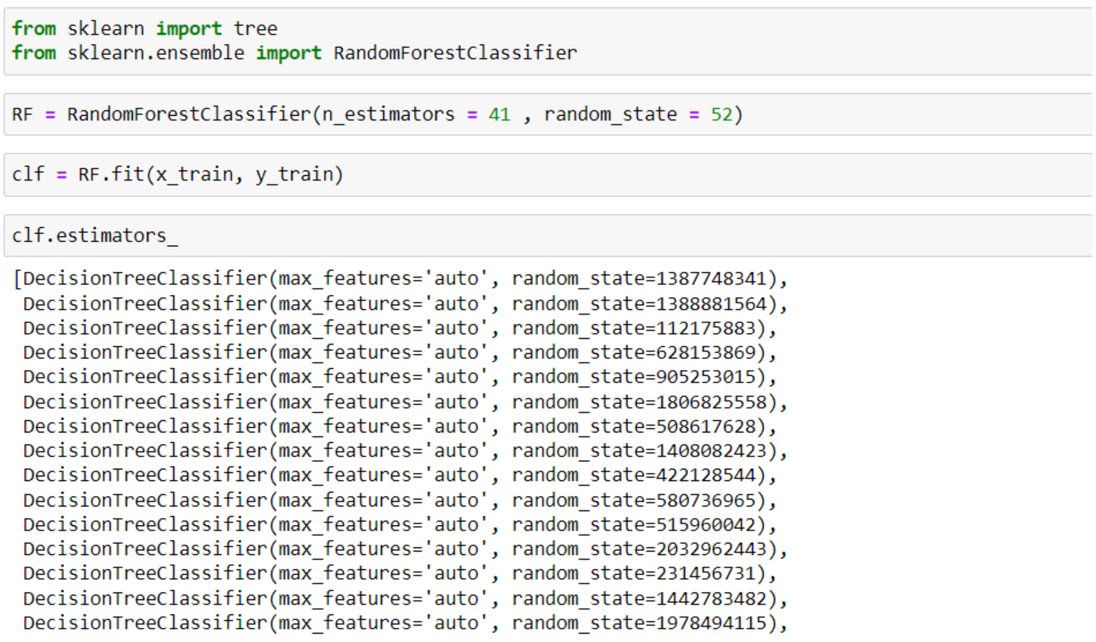
```
y_train.head()
```

```
802    19000
434    25000
900    18000
137    33000
413    22000
Name: Price, dtype: int64
```

Random Forest Classifier

Algorithm selection is a key challenge in any machine learning project since there is not an algorithm that is the best across all projects. Generally, we need to evaluate a set of potential candidates and select for further evaluation those that provide better performance.

In this project, I'm using Random Forest Classifier algorithms to implement in Scikit-Learn.



```
from sklearn import tree
from sklearn.ensemble import RandomForestClassifier
```

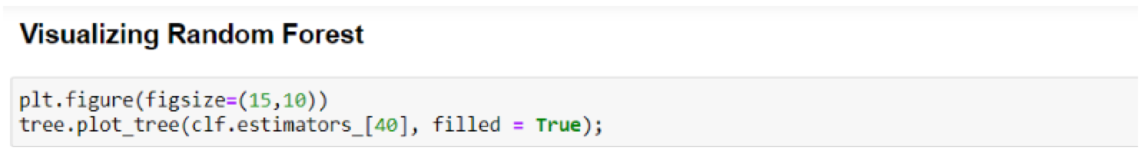
```
RF = RandomForestClassifier(n_estimators = 41 , random_state = 52)
```

```
clf = RF.fit(x_train, y_train)
```

```
clf.estimators_
```

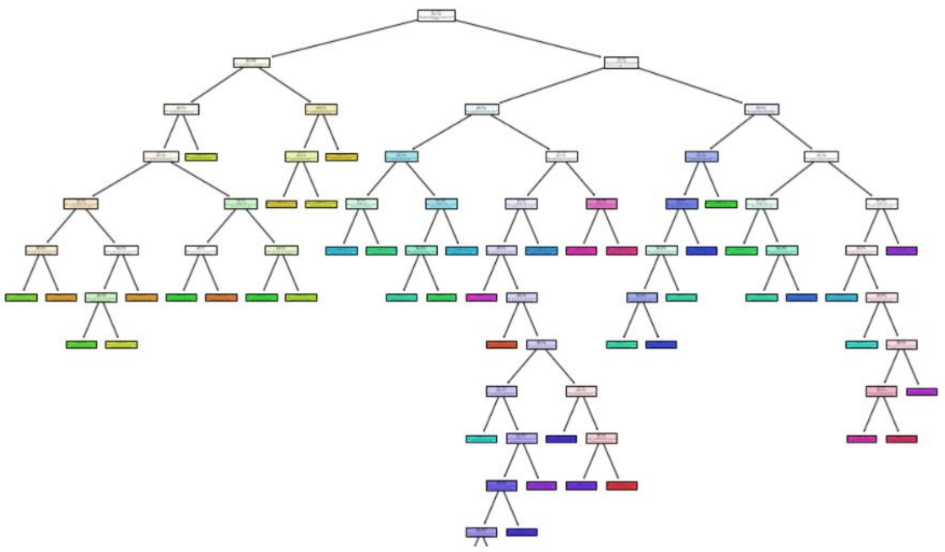
```
[DecisionTreeClassifier(max_features='auto', random_state=1387748341),
 DecisionTreeClassifier(max_features='auto', random_state=1388881564),
 DecisionTreeClassifier(max_features='auto', random_state=112175883),
 DecisionTreeClassifier(max_features='auto', random_state=628153869),
 DecisionTreeClassifier(max_features='auto', random_state=905253015),
 DecisionTreeClassifier(max_features='auto', random_state=1806825558),
 DecisionTreeClassifier(max_features='auto', random_state=508617628),
 DecisionTreeClassifier(max_features='auto', random_state=1408082423),
 DecisionTreeClassifier(max_features='auto', random_state=422128544),
 DecisionTreeClassifier(max_features='auto', random_state=580736965),
 DecisionTreeClassifier(max_features='auto', random_state=515960042),
 DecisionTreeClassifier(max_features='auto', random_state=2032962443),
 DecisionTreeClassifier(max_features='auto', random_state=231456731),
 DecisionTreeClassifier(max_features='auto', random_state=1442783482),
 DecisionTreeClassifier(max_features='auto', random_state=1978494115),
```

Visualization of Random Forest

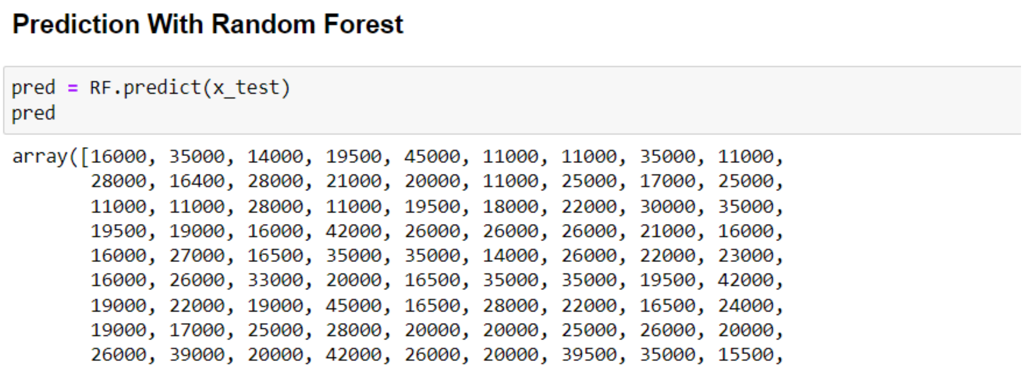


Visualizing Random Forest

```
plt.figure(figsize=(15,10))
tree.plot_tree(clf.estimators_[40], filled = True);
```



Evaluation metrices

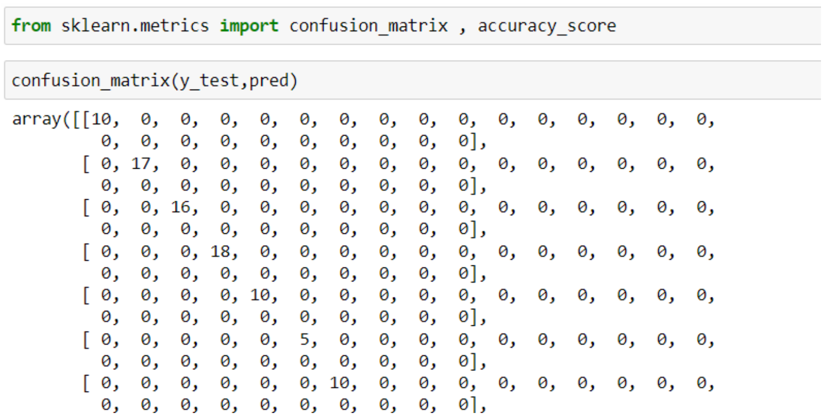


Prediction With Random Forest

```
pred = RF.predict(x_test)
pred
```

```
array([16000, 35000, 14000, 19500, 45000, 11000, 11000, 35000, 11000,
       28000, 16400, 28000, 21000, 20000, 11000, 25000, 17000, 25000,
       11000, 11000, 28000, 11000, 19500, 18000, 22000, 30000, 35000,
       19500, 19000, 16000, 42000, 26000, 26000, 26000, 21000, 16000,
       16000, 27000, 16500, 35000, 35000, 14000, 26000, 22000, 23000,
       16000, 26000, 33000, 20000, 16500, 35000, 35000, 19500, 42000,
       19000, 22000, 19000, 45000, 16500, 28000, 22000, 16500, 24000,
       19000, 17000, 25000, 28000, 20000, 20000, 25000, 26000, 20000,
       26000, 39000, 20000, 42000, 26000, 20000, 39500, 35000, 15500,
```

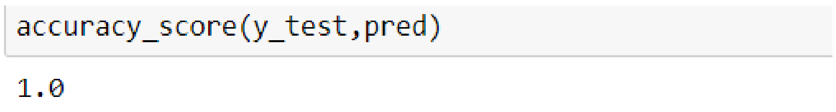
Evaluating the quality of the model is a fundamental part of the machine learning process. The most used performance evaluation metrics are calculated based on the elements of the confusion matrix.



```
from sklearn.metrics import confusion_matrix , accuracy_score
```

```
confusion_matrix(y_test,pred)
```

```
array([[10,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
         0,  0,  0,  0,  0,  0,  0,  0,  0,  0],
       [ 0, 17,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
         0,  0,  0,  0,  0,  0,  0,  0,  0,  0],
       [ 0,  0, 16,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
         0,  0,  0,  0,  0,  0,  0,  0,  0,  0],
       [ 0,  0,  0, 18,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
         0,  0,  0,  0,  0,  0,  0,  0,  0,  0],
       [ 0,  0,  0,  0, 10,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
         0,  0,  0,  0,  0,  0,  0,  0,  0,  0],
       [ 0,  0,  0,  0,  0,  5,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
         0,  0,  0,  0,  0,  0,  0,  0,  0,  0],
       [ 0,  0,  0,  0,  0,  0, 10,  0,  0,  0,  0,  0,  0,  0,  0,  0,
         0,  0,  0,  0,  0,  0,  0,  0,  0,  0],
```



```
accuracy_score(y_test,pred)
```

```
1.0
```

Conclusion

This model can be used to predict the house prices in any geographic location by just slightly fine tuning the features and parameters. And the accuracy obtain is 100%.

Predict whether the employee will leave the company or not based on left column

About the project

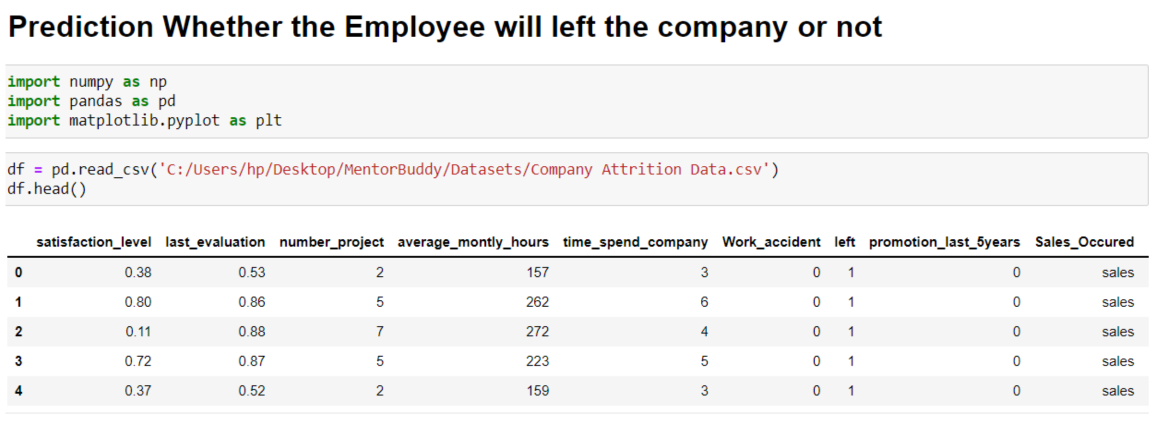
We know that larger companies contain more than thousand employees working for them, so taking care of the needs and satisfaction of each employee is a challenging task to do, it results in valuable and talented employees leave the company without giving the proper reason.

Employee churn is a major problem for many firms these days. Great talent is scarce, hard to keep and in high demand. Given the well-known direct relationship between happy employees and happy customers, it becomes of utmost importance to understand the drivers of employee dissatisfaction.

Understanding Data

Dataset used : Company Attrition Data

Importing Libraries and Dataset



Prediction Whether the Employee will left the company or not

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

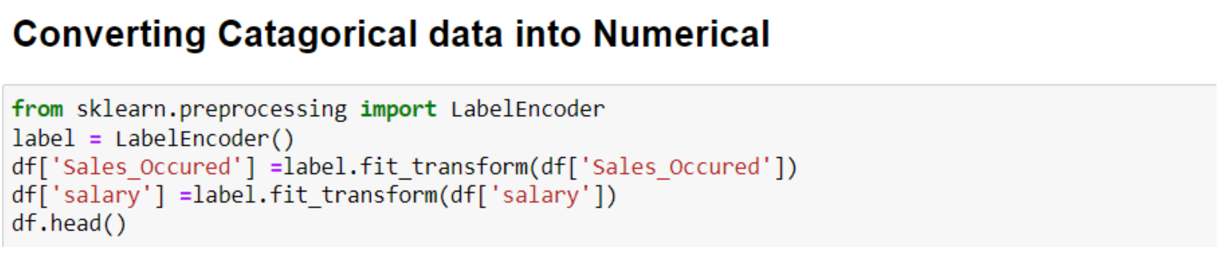
```python
df = pd.read_csv('C:/Users/hp/Desktop/MentorBuddy/Datasets/Company Attrition Data.csv')
df.head()
```

	satisfaction_level	last_evaluation	number_project	average_montly_hours	time_spend_company	Work_accident	left	promotion_last_5years	Sales_Occured
0	0.38	0.53	2	157	3	0	1	0	sales
1	0.80	0.86	5	262	6	0	1	0	sales
2	0.11	0.88	7	272	4	0	1	0	sales
3	0.72	0.87	5	223	5	0	1	0	sales
4	0.37	0.52	2	159	3	0	1	0	sales

The dataset contains 14999 rows and 10 columns

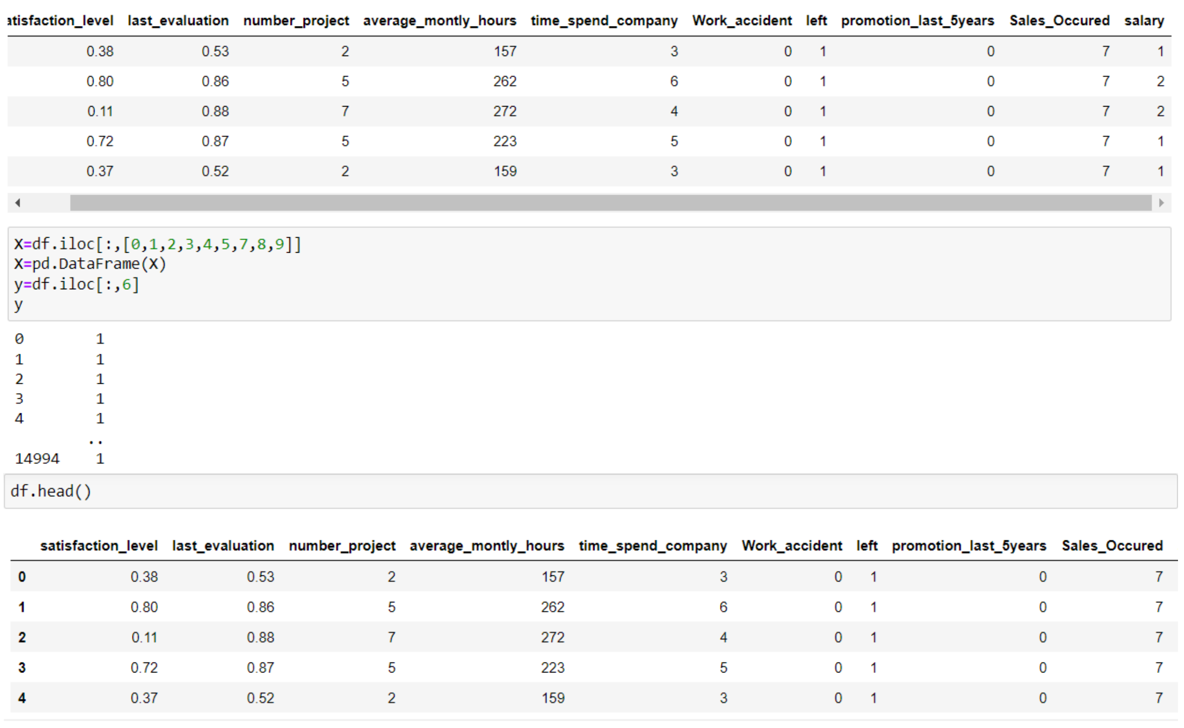
Data Pre-processing

The dataset has ‘salary’ and ‘sales’ column as categorical data, So we have to perform OneHotEncoding & LabelEncoding to convert this data into numerical form and To create dummy features we have to drop the first one to avoid linear dependency where some learning algorithms may struggle.



Converting Catagorical data into Numerical

```python
from sklearn.preprocessing import LabelEncoder
label = LabelEncoder()
df['Sales_Occured'] =label.fit_transform(df['Sales_Occured'])
df['salary'] =label.fit_transform(df['salary'])
df.head()
```



atisfaction_level	last_evaluation	number_project	average_montly_hours	time_spend_company	Work_accident	left	promotion_last_5years	Sales_Occured	salary
0.38	0.53	2	157	3	0	1	0	7	1
0.80	0.86	5	262	6	0	1	0	7	2
0.11	0.88	7	272	4	0	1	0	7	2
0.72	0.87	5	223	5	0	1	0	7	1
0.37	0.52	2	159	3	0	1	0	7	1

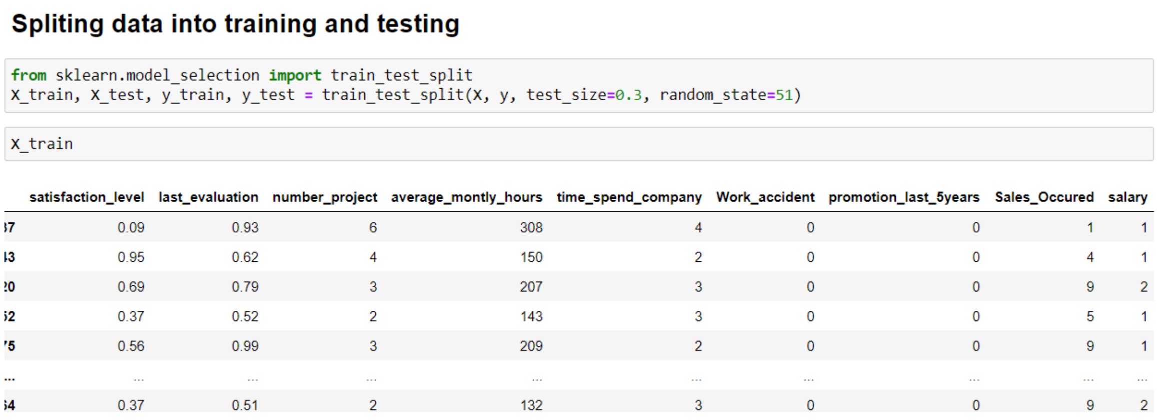
```
X=df.iloc[:,[0,1,2,3,4,5,7,8,9]]
X=pd.DataFrame(X)
y=df.iloc[:,6]
y
```

```
0        1
1        1
2        1
3        1
4        1
        ..
14994    1
```

```
df.head()
```

	satisfaction_level	last_evaluation	number_project	average_montly_hours	time_spend_company	Work_accident	left	promotion_last_5years	Sales_Occured
0	0.38	0.53	2	157	3	0	1	0	7
1	0.80	0.86	5	262	6	0	1	0	7
2	0.11	0.88	7	272	4	0	1	0	7
3	0.72	0.87	5	223	5	0	1	0	7
4	0.37	0.52	2	159	3	0	1	0	7

Splitting the data into training and testing



Spliting data into training and testing

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=51)
```

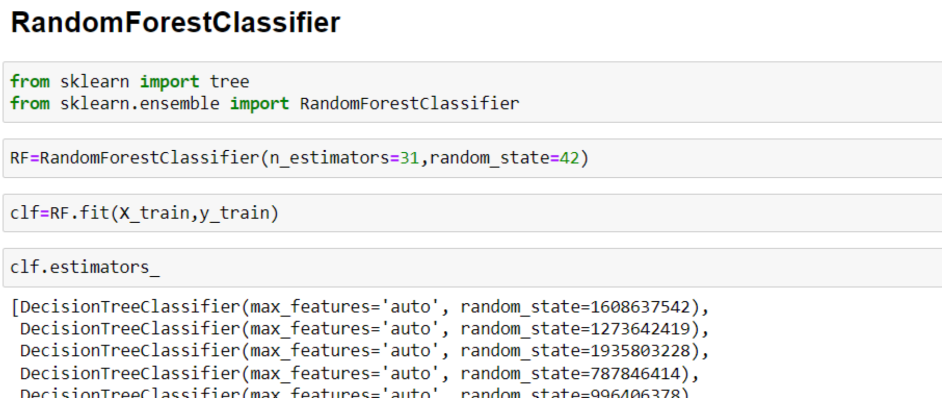
```
X_train
```

	satisfaction_level	last_evaluation	number_project	average_montly_hours	time_spend_company	Work_accident	promotion_last_5years	Sales_Occured	salary
37	0.09	0.93	6	308	4	0	0	1	1
43	0.95	0.62	4	150	2	0	0	4	1
20	0.69	0.79	3	207	3	0	0	9	2
52	0.37	0.52	2	143	3	0	0	5	1
75	0.56	0.99	3	209	2	0	0	9	1
...	...	...	...	...	...	...	...	...	...
54	0.37	0.51	2	132	3	0	0	9	2

Random Forest Classifier

Algorithm selection is a key challenge in any machine learning project since there is not an algorithm that is the best across all projects. Generally, we need to evaluate a set of potential candidates and select for further evaluation those that provide better performance.

In this project, I'm using Random Forest Classifier algorithms to implement in Scikit-Learn.



RandomForestClassifier

```
from sklearn import tree
from sklearn.ensemble import RandomForestClassifier
```

```
RF=RandomForestClassifier(n_estimators=31,random_state=42)
```

```
clf=RF.fit(X_train,y_train)
```

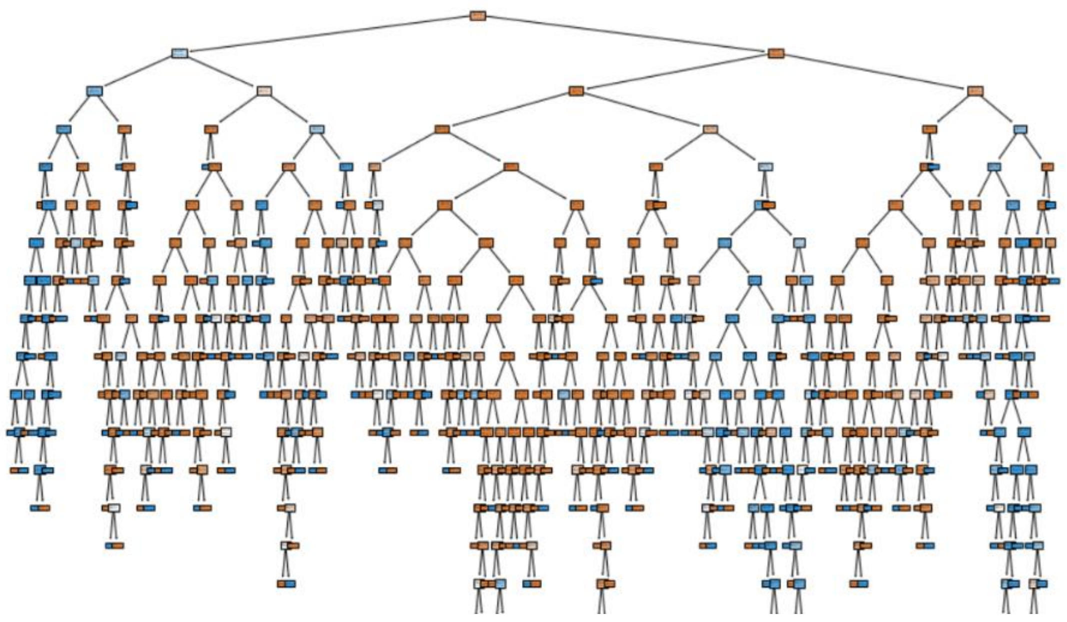
```
clf.estimators_
```

```
[DecisionTreeClassifier(max_features='auto', random_state=1608637542),
 DecisionTreeClassifier(max_features='auto', random_state=1273642419),
 DecisionTreeClassifier(max_features='auto', random_state=1935803228),
 DecisionTreeClassifier(max_features='auto', random_state=787846414),
 DecisionTreeClassifier(max_features='auto', random_state=996406378),
```

Visualization

Visualization

```
plt.figure(figsize=(15,10))
tree.plot_tree(clf.estimators_[0],filled=True);
```



```
pred=RF.predict(X_test)
pred
```

```
array([0, 1, 0, ..., 0, 1, 0], dtype=int64)
```

Evaluation Metrics

Evaluating the quality of the model is a fundamental part of the machine learning process. The most used performance evaluation metrics are calculated based on the elements of the confusion matrix.

Predicting Accuracy

```
from sklearn.metrics import confusion_matrix, accuracy_score
```

```
confusion_matrix(y_test,pred)
accuracy_score(y_test,pred)*100
```

```
99.0
```

Reason for leaving the company

Let’s see what are the reason that causing employees to leave the company and what are the factors behind it.

Reason For Leaving The Company

```
Reason1=pd.DataFrame()
i=0
WithorNotWorkAccident=0
while i<len(y):
        if(df.iloc[[i],0].values> 0.5 and df.iloc[[i],1].values > 0.5 and df.iloc[[i],4].values>
            a=[df.iloc[i,[7,9]].values]
            df1=pd.DataFrame(a)
            Reason1=Reason1.append(df1)
            i=i+1
            WithorNotWorkAccident=WithorNotWorkAccident+1
        else:
            i=i+1

Reason1=Reason1.reset_index()
Reason1=Reason1.drop('index',1)
```

```
Reason2=pd.DataFrame()
i=0
WithWorkAccident=0
while i<len(y):
        if(df.iloc[[i],0].values> 0.5 and df.iloc[[i],1].values > 0.5 and df.iloc[[i],4].values> 4 and df.iloc[[i],5].values==1)
            a=[df.iloc[i,[7,9]].values]
            df1=pd.DataFrame(a)
            Reason2=Reason2.append(df1)
            i=i+1
            WithWorkAccident=WithWorkAccident+1
        else:
            i=i+1

Reason2=Reason2.reset_index()
Reason2=Reason2.drop('index',1)

Reason2WithoutAcc=pd.DataFrame()
i=0
WithoutWorkAccident=0
while i<len(y):
        if(df.iloc[[i],0].values> 0.5 and df.iloc[[i],1].values > 0.5 and df.iloc[[i],4].values> 4 and df.iloc[[i],5].values==0)
            a=[df.iloc[i,[7,9]].values]
            df1=pd.DataFrame(a)
            Reason2WithoutAcc=Reason2WithoutAcc.append(df1)
            i=i+1
            WithoutWorkAccident=WithoutWorkAccident+1
        else:
            i=i+1

Reason2WithoutAcc=Reason2WithoutAcc.reset_index()
Reason2WithoutAcc=Reason2WithoutAcc.drop('index',1)
```

```
Reason3=pd.DataFrame()
i=0
WithHigherProjects=0
while i<len(y):
        if(df.iloc[[i],0].values> 0.5 and df.iloc[[i],1].values > 0.5 and df.iloc[[i],4].values> 4 and df.iloc[[i],2].values>3):
            a=[df.iloc[i,[7,8,9]].values]
            df1=pd.DataFrame(a)
            Reason3=Reason3.append(df1)
            i=i+1
            WithHigherProjects=WithHigherProjects+1
        else:
            i=i+1

Reason3=Reason3.reset_index()
Reason3=Reason3.drop('index',1)
```

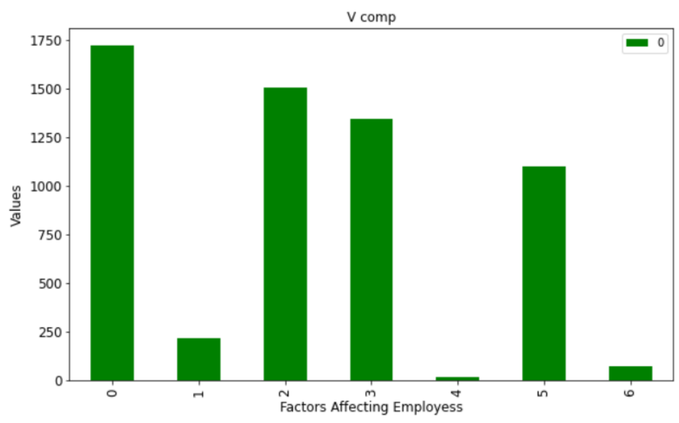
```
Reason4=pd.DataFrame()
i=0
WithPromotion=0
while i<len(y):
        if(df.iloc[[i],0].values> 0.5 and df.iloc[[i],1].values > 0.5 and df.iloc[[i],4].values> 4 and df.iloc[[i],2].values>3 a
            a=[df.iloc[i,[9]].values]
            df1=pd.DataFrame(a)
            Reason4=Reason4.append(df1)
            i=i+1
            WithPromotion=WithPromotion+1
        else:
            i=i+1

Reason4=Reason4.reset_index()
Reason4=Reason4.drop('index',1)
```

```
Reason5=pd.DataFrame()
i=0
WithHigherTime=0
while i<len(y):
        if(df.iloc[[i],0].values> 0.5 and df.iloc[[i],1].values > 0.5 and df.iloc[[i],4].values> 4 and df.iloc[[i],2].values>3 ar
            a=[df.iloc[i,[9]].values]
            df1=pd.DataFrame(a)
            Reason5=Reason5.append(df1)
            i=i+1
            WithHigherTime=WithHigherTime+1
        else:
            i=i+1

Reason5=Reason5.reset_index()
Reason5=Reason5.drop('index',1)
```

```
Reason6=pd.DataFrame()
i=0
WithLowerSalary=0
while i<len(y):
        if(df.iloc[[i],0].values> 0.5 and df.iloc[[i],1].values > 0.5 and df.iloc[[i],4].values> 4 and df.iloc[[i],2].values>3 ar
            a=[df.iloc[i,[7]].values]
            df1=pd.DataFrame(a)
            Reason6=Reason6.append(df1)
            i=i+1
            WithLowerSalary=WithLowerSalary+1
        else:
            i=i+1

Reason6=Reason6.reset_index()
Reason6=Reason6.drop('index',1)
```

Comparisson

```
b=[WithorNotWorkAccident,WithWorkAccident,WithoutWorkAccident,WithHigherProjects,WithPromotion,WithHigherTime,WithLowerSalary]
Visual=pd.DataFrame(b)

ax = Visual[0].plot(kind='bar', title ="V comp",figsize=(10,6), color="green", legend=True, fontsize=12)
ax.set_xlabel("Factors Affecting Employess", fontsize=12)
ax.set_ylabel("Values", fontsize=12)
plt.show()
```



Conclusion

Here in the above graph numbers on x-axis from 0 to 6 are representing WithHigherProjects,WithLowerSalary,WithHigherTime,WithPromotion,WithWorkAccident,WithorNotWorkAccident,WithoutWorkAccident. Each of these are the factors which can affect employment as WithHigherTime represents, employees who have more than four year of work experience but still haven’t got any promotion is 1750 which is a significant amount, WithLowerSalary represents employees whose salary level is low even when their evaluation score was higher than 3 such employees are 750.

Thus, after evaluating this dataset, we get to know that lower salary levels, no promotions even when employees are working more than 4 years are the two main reasons for the employees to leave the organization.

This model predict the accuracy with 99%.